ABSTRACT

Title of dissertation: SOLVING, GENERATING, AND MODELING ARC ROUTING PROBLEMS

Oliver Lum, Doctor of Philosophy, 2017

Dissertation directed by: Professor Bruce Golden
Department of Decision, Operations, and Information Technology

Arc routing problems are an important class of network optimization problems. In this dissertation, we develop an open source library with solvers that can be applied to several uncapacitated arc routing problems. The library has a flexible architecture and the ability to visualize real-world street networks. We also develop a software tool that allows users to generate arc routing instances directly from an open source map database. Our tool has a visualization capability that can produce images of routes overlaid on a specific instance.

We model and solve two variants of the standard arc routing problem: (1) the windy rural postman problem with zigzag time windows and (2) the min-max K windy rural postman problem. In the first variant, we allow servicing of both sides of some streets in a network, that is, a vehicle can service a street by zigzagging. We combine insertion and local search techniques to produce high-quality solutions to

a set of test instances. In the second variant, we design a cluster-first, route-second heuristic that compares favorably to an existing heuristic and produces routes that are intuitively appealing. Finally, we show how to partition a street network into routes that are compact, balanced, and visually appealing.

SOLVING, GENERATING, AND MODELING ARC ROUTING PROBLEMS

by

Oliver Lum

Dissertation submitted to the Faculty of the Graduate School of the
University of Maryland, College Park in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
2017

Advisory Committee:
Professor Bruce Golden, Chair/Advisor
Professor Denny Gulick
Professor Ilya Ryzhov
Professor Paul Schonfeld
Professor Edward Wasil, Co-Advisor

© Copyright by
Oliver Lum
2017

Dedication

Dedicated to Dr. Bertram Hui, Dr. Benjamin Ewy, Dr. Mari Maeda, Mr. Doran Michels, Dr. Vincent Tang, Dr. John Paschkewitz, and Dr. Mark Wrobel for taking a chance on an unremarkable student with little direction or ambition and showing him you don't have to choose between the blue skies and making a difference.

Acknowledgments

I owe my gratitude to all the people who have made this dissertation possible. First, I'd like to thank my advisor Professor Bruce Golden for his tremendous patience and invaluable guidance. His support and encouragement throughout the last five years cannot be overstated, and it has truly been a pleasure to study under his tutelage. I consider it a great privilege to have had the opportunity to work with him.

I would also like to thank my co-advisor Professor Edward Wasil for his clarity of thought and many, many hours spent helping me improve my own. Thanks to Professor Ángel Corberán for the opportunity to collaborate with him on some of the work in Chapter 6 of this volume. Finally, I would like to thank Professor Ilya Ryzhov, Professor Denny Gulick, and Professor Paul Schonfeld for donating their time to serve on my committee and for sparing their invaluable time reviewing the manuscript.

Of course, my deepest gratitude belongs to my close friends and family without whom I surely would not have had the willpower or determination to complete my studies. To my mother Shirley, my father John, and my brother Alan: thank you for your unconditional love, and tremendous poise in accommodating my hectic and occasionally difficult lifestyle. I look forward to no longer having to rush through our lunches to get home and work. To my oldest friends Andrew Campbell, Mollie Semmes, and Asha and Monica Saavoss: words cannot express how grateful I am to have had your constant companionship and camaraderie throughout the years.

To Shirley Pon (and Momo): your aggressive ebullience is perpetual motivation. I shall be eternally grateful for showing me that the depth of a friendship is cultivated by a mutual willingness to be open. To David Moser: thank you for teaching and encouraging me to share my appreciation for the beauty in everyone and everything. Finally, to Polina Sitnova: your passion for kindness has enriched my life in more ways than you will ever likely know. Thank you for teaching me a happiness not tied to accomplishment, or achievement, but to the happiness and welfare of a kindred soul. That freedom has been invaluable to weathering the last five years with relative ease.

I would also like to acknowledge all of my colleagues at Strategic Analysis (SA) and the Defense Advanced Research Projects Agency (DARPA) for employing me and for their understanding, allowing me to maintain a flexible work schedule during the last 3 and a half years. The unique opportunity to work with these outstanding individuals on tremendously interesting projects has provided me a perspective that has been transformational.

Finally, I would like to thank the fellow members of my research group for providing their input and advice throughout my tenure as a graduate student, especially Dr. Rui Zhang for his help with the IP formulation in Chapter 4, and Dr. Xingyin Wang for sharing his knowledge about navigating the graduate school experience.

Table of Contents

List of Tables

List of Figures

List of Abbreviations

API	Application Programming Interface
ARP	Arc Routing Problem
ATD	Average Task Distance
BNC	Branch and Cut
CI	Connectivity Index
CPP	Chinese Postman Problem
CSS	Cascading Style Sheets
D3	Data-Driven Documents
DCPP	Directed Chinese Postman Problem
DRPP	Directed Rural Postman Problem
GB	Gigabyte
GIS	Geographic Information System
HO	Hull Overlap
HTML	Hypertext Markup Language
IP	Integer Program
JS	Javascript
JSON	Javascript Object Notation
MCPP	Mixed Chinese Postman Problem
MIP	Mixed Integer Program
MMKWRPP	Min-Max K Windy Rural Postman Problem
NO	Node Overlap
OAR Lib	Open Source Arc Routing Library
OAR Bench	Open Source Arc Routing Benchmarking Utility
OSM	Open Street Maps
PFIH	Push Forward Insertion Heuristic
ROI	Route Overlap Index
SAPH	Shortest Additional Paths Heuristic
UCPP	Undirected Chinese Postman Problem
UI	User Interface
VRP	Vehicle Routing Problem
VRP Bench	Vehicle Routing Problem Benchmarking Utility
VRPTW	Vehicle Routing Problem With Time Windows
WPP	Windy Postman Problem
WRPP	Windy Rural Postman Problem
WRPPZTW	Windy Rural Postman Problem with Time Dependent Zigzag Service

Chapter 1: Introduction

Businesses and organizations that conduct logistics operations or provide services often must schedule and plan deliveries to customers. This involves coordinating a fleet of vehicles traversing street networks to reach those customers. Problems like this arise in a variety of domains including industry, government, and military applications. For example, FedEx, UPS, and Amazon must determine which customers will be served by which distribution center, and provide each delivery vehicle with an ordering of the residences and businesses that receive packages each day. Municipalities must determine which streets should be plowed first in the event of significant snowfall, and then plan how to assign the streets to vehicles for plowing. The military must decide how to conduct patrols that provide sufficient coverage of an area while avoiding predictable repetition of routes in order to mitigate the risk of an attack. Although these examples have different goals (minimize monetary cost, time, or predictability), they all involve the solution of a combinatorial optimization problem on a network.

Arc Routing Problems (ARPs) involve constructing a set of paths in a network that minimize cost, where cost can incorporate time, money, customer satisfaction, and many other real-world features. Historically, the first ARP was the Bridges

of Königsberg problem formulated by Euler in 1741 [1]. Modern study of ARPs began with the Chinese postman problem (CPP) posed by Guan in 1962 [2]. The CPP models a mail carrier who delivers mail to customers with locations along the streets of a neighborhood. In this problem, each street has a known traversal cost. The objective is to create a single route that traverses every street in the network and begins and ends at a starting location (depot). For networks that contain only one-way streets, or only two-way streets, this problem can be solved quickly. However, for networks that contain both, or that incorporate complicating factors (e.g., customer time windows, turn penalties, asymmetric travel costs), the problem is provably intractable.

In this dissertation, we develop two software tools designed to help the research community test and solve ARPs more quickly and effectively. We also create heuristics for several arc routing variants and discuss alternative metrics to assess the quality of routes including one metric that we develop.

The first contribution of this dissertation is the creation of an open source arc routing library (OAR Lib). OAR Lib contains Java implementations of solvers and heuristics for many of the standard arc routing problems found in the literature. More complex variants can be formulated by including problem features such as multiple vehicles and alternative modes of service. Heuristics for the more complex variants often include solving the standard problems as subproblems during the algorithm. OAR Lib allows researchers to use the solvers directly. An application programming interface can be used to integrate the solvers into other software. OAR Lib is freely available to the research community at https://github.com/

Olibear/ArcRoutingLibrary. We discuss the contents of the library and validate our implementation using data found in the literature.

Second, we develop an open source arc routing benchmark instance generator (OAR Bench). When researchers develop a new problem formulation, optimal procedure, or heuristic, they usually perform computational tests on a set of instances. The computational results demonstrate the performance of the algorithm in terms of run time, scalability, and solution quality. Ideally, these instances should reflect the real-world street networks encountered in practice. OAR Bench is a geographic information system that allows users to generate test instances for ARPs using real-world street data. These data include the geographic position of streets, information about whether a street is a main thoroughfare, speed limit, and types of buildings along the street. OAR Bench is freely available to the research community and the source code is open at https://github.com/Olibear/ArcRoutingBenchmark.

Third, we develop a heuristic for the Windy Rural Postman Problem with Time Dependent Zigzag Service (WRPPZTW). In the WRPPZTW, the graph has asymmetric travel costs (windy) and a subset of the streets that require service (rural). It is possible to perform zigzag service on a subset of the required streets to service customers on both sides of the street simultaneously. Furthermore, there are time restrictions on when the zigzag operation can be performed. We focus on instances of the WRPPZTW where zigzagging can be done from the beginning of the planning period until some known time T. This models the situation where light traffic volume in the early morning allows vehicles to perform zigzag service. We develop a heuristic that uses an insertion procedure to solve for the early part

of the route (until approximately time T), and the solves an integer program (IP) to complete the route. The IP is a new formulation for the Windy Rural Postman Problem with Zigzag Service (WRPPZ). We present computational results and compare performance with an existing solver for the WRPPZTW.

Fourth, we create a cluster-first, route-second heuristic for the Min-Max K Windy Rural Postman Problem (MMKWRPP). We introduce a new route metric that measures the degree to which a set of routes is non-overlapping. In the MMKWRPP, the task is to route a fleet of K homogeneous vehicles to service a set of customers in a windy network. The cost of the routes is equal to the length of the longest route. This objective function incorporates route balance and route length considerations. For ARPs that involve a fleet of vehicles, practitioners may want routes to be compact and non-overlapping. Our heuristic partitions the graph into areas of coverage for each vehicle in the fleet. We present computational results and compare performance with an existing heuristic for the MMKWRPP. Our heuristic produces comparable results with respect to the min-max objective, and performs favorably with respect to metrics that measure compactness and route overlap. We introduce a new metric (hull overlap) to evaluate the aesthetic quality of routes. We incorporate two aesthetic measures into the integer programming formulation of the MMKWRPP. We produce Pareto fronts for small instances to understand the tradeoff between the min-max objective and the aesthetic metrics.

The dissertation is organized as follows. In Chapter 2, we describe the Chinese Postman Problem and the Rural Postman Problem and discuss our implementation of solution techniques and heuristics that are included in OAR Lib. In Chapter 3, we

describe the process of benchmarking solution procedures and introduce OAR Bench to generate realistic test instances for arc routing problems. Chapters 4 through 6 present heuristics for two arc routing problems. In Chapter 4, we describe the WRPPZTW and develop a heuristic that produces solutions that compare favorably to those produced by an existing solution procedure. In Chapter 5, we introduce the MMKWRPP and survey the literature on aesthetic route quality metrics. We develop a cluster-first, route-second heuristic and compare the solutions it produces to an existing heuristic for the MMKWRPP with respect to cost and three aesthetic measures. In Chapter 6, we investigate incorporating two aesthetic metrics into the mathematical formulation of the MMKWRPP in a multiobjective setting. Chapter 7 briefly reviews our contributions and presents our conclusions.

Chapter 2: OAR Lib: An Open Source Arc Routing Library

2.1 Introduction

In vehicle routing, problem complexity (e.g., costs, constraints such as time windows, etc.) is associated with the nodes and the edges of the underlying street network. Problem variants where vehicles service customers at the nodes are referred to as node routing problems. These include the traveling salesman problem and the vehicle routing problem. Variants where customers lie on the edges are referred to as arc routing problems. In both cases, the goal is to minimize an objective function that reflects the total cost of the routes.

Nearly all routing problems have been shown to be NP-Hard, so it is unlikely that they can be solved to optimality in a computationally tractable manner [3]. Heuristics avoid this intractability by finding very good solutions that are nearly optimal. Heuristics are evaluated based on efficiency and accuracy.

Different implementations of the same algorithm can lead to different results, but this variability is difficult to control. For example, differences in data structures and how memory is managed can drastically affect the runtime required to solve a particular instance. The effect of these differences is more apparent when these heuristics are used as subroutines. For example, suppose there are two competing

metaheuristics A and B. Both solve a shortest paths problem during their respective initialization procedures. Since the researcher responsible for A uses a more efficient shortest paths algorithm, significantly faster runtimes are reported. However, if this difference were eliminated, A would be slower than B. One would reasonably, but erroneously, conclude that metaheuristic A is superior. Therefore, it is important to standardize solvers so that fair comparisons can be made and the merits of an algorithm can be attributed solely to its design.

We develop an open source code library that provides a set of standard solvers for arc routing problems. The library contains solvers for the following problems: the Chinese Postman Problem on a directed graph (DCPP), the CPP on an undirected graph with symmetric traversal costs (UCPP), the CPP on a mixed graph (MCPP), the CPP on an undirected graph with directionally asymmetric costs (WPP for Windy Postman Problem), and the Rural Postman Problem (RPP) on directed (DRPP) and windy graphs (WRPP) where not all arcs are required to be traversed in the solution. For each problem, if it is not possible to efficiently solve it to optimality, we implement a well-known heuristic in our library. If the problem is solvable in polynomial time, we implement the exact algorithm in our library. For the details of each specific algorithm, consult references for the DCPP, UCPP, MCPP, WPP, DRPP, and WRPP [4–11]. In Table 2.1, we summarize the problems that are addressed in the library. In Table 2.2, we summarize the performance of the heuristics contained in the library. We point out that the paper by Groër et al. [12] complements our work and presents a library of local search heuristics for node routing problems.

Problem	Required Links	Exact Solver	Heuristics	References
Directed Chinese Postman Problem	All	1	0	[5]
Undirected Chinese Postman Problem	All	1	0	[5]
Mixed Chinese Postman Problem	All	0	2	[6, 7]
Windy Postman Problem	All	0	2	[8, 11]
Directed Rural Postman Problem	Subset	0	1	[10]
Windy Rural Postman Problem	Subset	0	1	[11]

Table 2.1: A summary of the problems addressed in the library. The required links column shows whether all links in the graph require traversal, or only a subset require traversal. The exact solver column shows the problems that can be solved exactly in the library. The heuristics column shows the number of heuristics in the library for each problem. The references column provides the original source of the algorithms.

2.2 Definitions

A graph $G = (V, L)$ where V is a set of vertices (also referred to here as nodes) and L is a set of links. Vertices are defined by v_i and links are represented as ordered pairs $e_{ij} = (i, j)$ where both v_i and v_j are members of the vertex set. A link $l = (i, j)$ also has a traversal cost c_{ij}. A link is an edge if it is undirected (it can be traversed from i to j and from j to i). A link is an arc if it is directed (it can only be traversed from i to j). In the case of arcs, the first element of the ordered pair is referred to as the tail, while the second is referred to as the head. In an undirected graph, all members of the link set are edges and the graph is denoted by (V, E). In a directed

Solver	Average Deviation (%)	Max Deviation (%)	Benchmark Instances	Reference
Frederickson's heuristic for the MCPP	15.9	78.0	Manhattan, Complete, and Special Case	[7]
Yaoyuenyong et al.'s heuristic for the MCPP	1.0	14.4	Manhattan, Complete, and Special Case	[7]
Christofides's heuristic for the DRPP	24.23	64.11	Campos and Savall	[10]
Benavent et al.'s heuristic for the WRPP	4.17	Not provided	Albaida-Madrigueras	[11]

Table 2.2: A computational summary of the heuristics in the library. Deviations are percentages from lower bounds presented in the cited reference. The benchmark instance column lists Note that the references in this table are to the papers which provide the computational results listed here. For the references which introduce the heuristics, consult Table 2.1.

graph, all members of the link set are arcs, and the graph is denoted by (V, A). A mixed graph has both types of links and is denoted by (V, E, A). A windy graph is undirected with asymmetric traversal costs (the cost of going from vertex i to vertex j may not be the same as the cost of going from vertex j to vertex i).

A windy graph can model an undirected graph, a directed graph, and a mixed graph. For an undirected graph, set $c_{ij} = c_{ji}\ \forall i, j$; for a directed graph, set $c_{ij} = c_a$ and $c_{ji} = N\ \forall a = (i, j) \in A$ where N is a very large value, much greater than $\sum_{a \in A} c_a$; for a mixed graph, it follows directly from the previous two graph types. Thus, any solution method that can be applied to a problem on a windy graph can also be applied to the same problem on the three types of graphs.

A graph is strongly connected if it is possible to reach any vertex from any

other vertex. For any pair of vertices i and j, it is possible to construct an ordered list of links $(i_0, j_0), (j_0, j_1), (j_1, j_2), ..., (j_{k-1}, j_k)$ where $i_0 = i$ and $j_k = j$. An ordered list of this form is called a path. A path with minimal traversal cost in the graph is known as a shortest path. We denote the cost of a shortest path between vertex i and vertex j as sp_{ij}.

A circuit is defined as a path that begins and ends at the same vertex. A graph is Eulerian if and only if there exists an Eulerian circuit, (a circuit that traverses every link in the graph exactly once). The following is a list of well-known conditions for a graph to be Eulerian.

- *Undirected*: An undirected graph is Eulerian if and only if every node has even degree (a property known as evenness).
- *Directed*: A directed graph is Eulerian if and only if the in-degree equals the out-degree for every node (a property known as symmetry).
- *Mixed*: A mixed graph is Eulerian if and only if every node has even degree and the graph is balanced. For any subset S of V, |number of arcs from S to $V \setminus S$ - number of arcs from $V \setminus S$ to S| $\leq$ number of edges from S to $V \setminus S$. A sufficient condition is that every node has even degree, and the in-degree equals the out-degree.

A graph $G_2 = (V_2, L_2)$ is an augmentation of the graph $G_1 = (V_1, L_1)$ if $V_1 \subseteq V_2$, $L_1 \subseteq L_2$, and $\forall l^2_{ij} \in L_2, (\exists\ l^1_{ij} \in L_1 \quad \text{and} \quad cost(l^2_{ij}) = cost(l^1_{ij}))$. Every link in the original graph appears in the augmentation. The augmentation only includes copies of links in the original graph.

In an undirected graph, the degree of a vertex is simply the number of edges incident to the vertex. In a directed graph, the in-degree of a vertex v is the number of arcs $a \in A$ for which v is the head, and the out-degree of a vertex v is the number of arcs for which v is the tail. For a mixed graph, our definition of in-degree and out-degree remain the same. The two properties only take into account arcs in the graph, while our definition of degree only includes edges as though the arcs were deleted from the graph.

The problems we consider fall under the following two categories.

- *Chinese Postman Problem:* Given a graph G (either directed, undirected, mixed, or windy) and a cost function c_{ij} (associated with traversing the link (i, j)), find a circuit that traverses each link (edge or arc) at least once and minimizes the total traversal cost.

- *Rural Postman Problem:* Given a graph G (either directed or windy), a set of required links $L_R \subseteq L$, a depot location (one of the vertices), and cost function c_{ij} (associated with traversing the link (i, j)), find a circuit beginning and ending at the depot that traverses each required arc at least once and minimizes the total traversal cost.

In both the CPP and the RPP, the problem is solved in two steps. First, find an Eulerian augmentation of the original graph. Second, find the Eulerian circuit in the augmented graph. The second step can be performed easily and efficiently in linear time using Hierholzer's algorithm [13].

We use the following common notation.

- $\delta(v)$ = in-degree minus the out-degree,
- D^+ and D^- are the set of vertices with $\delta(v) > 0$ and $\delta(v) < 0$,
- $\mathbb{Z}_+^0$ is the set of non-negative integers ($\mathbb{N}$).

2.3 Features of the Library

We now describe some of the features in our library.

In order to depict networks and the corresponding vehicle routes, we use Gephi [14]. Gephi is an open source visualization utility with an application programming interface (API) that enables quick integration. Gephi allows several layout routines that attempt to minmize visual clutter while preserving distance relationships.

The library contains the ability to manually specify vertex coordinates. We use these subroutines to export any graph created within the library to a portable document format (pdf) file.

To allow the quick, dynamic generation of new test instances, we have leveraged the queryable Open Street Maps (OSM) [15] database to allow a user with an Internet connection to specify a geographic bounding box and retrieve the associated street network. The largest contiguous subgraph is returned so that connectivity is ensured. Figure 2.1 gives a street network returned by OSM sampling a portion of the street network in Helsinki, Finland.

We have decoupled seven graph algorithms that are used in multiple solvers to allow for them to be used in contexts not originally coded by the authors, including inside more sophisticated solvers.

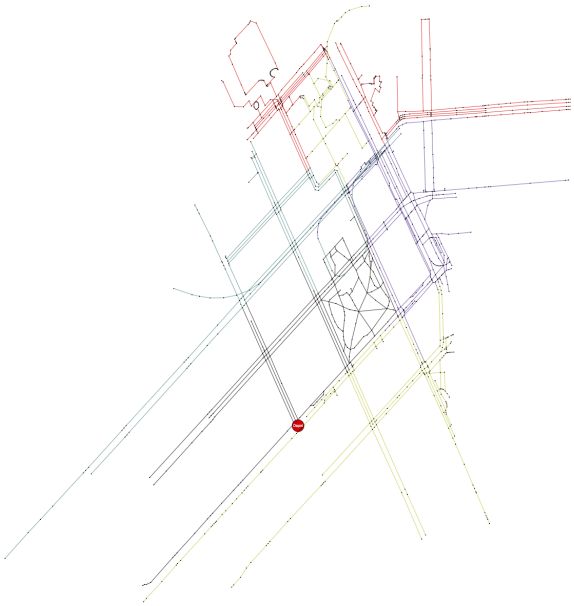


Figure 2.1: A graph obtained using OSM by querying the street network of Helsinki, Finland and exported through the Gephi integration.

- Floyd-Warshall all-pairs shortest paths algorithm [16]
- Dijkstra's single-source shortest paths algorithm with priority queues [17]
- Successive shortest paths min cost flow algorithm [18]
- Blossom V min cost matching algorithm [19]
- Hierholzer's algorithm [13]
- Minimum spanning tree algorithm [20]
- Strongly connected components algorithm [21]

2.4 Architecture

We discuss the structure of the library and outline how the core architecture can be extended to implement various solvers.

The library's core package contains the abstractions upon which the rest of the library is based. The graph, link, and vertex abstractions maintain a unique (per type) global identifier (ID). This ID may be used to compare elements that may otherwise appear similar (e.g., when manipulating multiple graphs, both depots may be referred to as v_1 but the global ID property will distinguish between the two). In addition, links and vertices have a local ID. The local ID is based on the graph that the node or link is assigned to. Therefore, the first vertex assigned to each graph will have a local ID of one. Finally, links and vertices may keep track of a match ID. The match ID may be set to allow the correspondence between elements in a graph. For example, the solution of a flow problem on an auxiliary graph can be used to construct an augmentation of the original graph. In this case, the match ID of edges in the original graph can be set to the local IDs of edges in the auxiliary graph to allow a user to keep track of the correspondence.

Graph objects maintain additional state information. The shortest path matrix is available on demand, and storage is implemented according to a lazy design pattern (the memory is not allocated until it is first needed, and only recomputed if the state of the graph changes). The (local) ID of the depot and the element sets (V and E) are maintained by the graph. Functions for standardized creation of vertices and links, as well as getters (methods which allow other objects to retrieve a specific vertex or edge) are provided. Finally, a method for providing deep copy (a distinct copy as opposed to a reference to the existing object) is specified by all graphs.

The problem and solver abstractions are included in the core package. Prob-

lem instances contain the underlying network, objective function, and any problem features such as time windows and additional service costs. We can restrict the ability to restrict the applicability of the solver to only those networks with certain structure via the checkGraphRequirements method. This feature enables a solver to specify the conditions for a feasible solution (e.g., strong connectedness for solving the DCPP).

Improvement procedures are also implemented as an abstract class that allows for composition of improvement frameworks, as well as single moves. This creates an opportunity for both individual procedures (e.g., the arc routing analog of a 2opt) and frameworks (i.e., an ordered set of procedures with termination criteria) to be modular when tuning a heuristic.

The mover object simplifies the accounting associated with writing improvement procedures. It performs swaps for arc routing problems by manipulating routes as ordered lists of required links (with shortest paths assumed in between), and interchanging positions of the links in the lists. This compact representation of a route was first proposed by Benavent et al. in [11]. It allows for many improvement procedures from node routing to be used in an arc routing setting.

Wherever possible, we use fast and memory efficient data structures to organize elements of the graph. For example, we use Trove [22] which provides memory-optimized variants of several native Java data structures to store vertices and edges. Elements are usually referenced via their local ID. Alternative sparse matrix representations of the graph structure can have increased storage requirements, and are difficult to adapt to graphs with multiple edges between v_i and v_j. These are needed

Solver	Description
Edmonds and Johnson's Algorithm for the DCPP [4] (Exact)	Solve a min cost flow problem. Symmetry is achieved by adding arcs according to the solution to the flow problem.
Edmonds's Algorithm for the UCPP [5] (Exact)	Solve a min cost matching problem. Evenness is achieved by adding edges according to the solution.
Frederickson's Heuristic for the MCPP [6]	Use Edmonds's algorithms for the UCPP and DCPP separately to achieve each property. Repair the augmentation by eliminating special circuits in the graph.
Yaoyuenyong et al.'s Heuristic for the MCPP [7]	Apply augmentation procedures from Frederickson's heuristic. Apply improvement procedures that search for opportunities to replace added links with shortest paths.
Win's Heuristic for the WPP [8]	Create an auxiliary undirected graph with $c_{ij}^{aux} = .5 * (c_{ij} + c_{ji})$. Solve the UCPP on the auxiliary graph. Augment the original graph according to the UCPP solution on the auxiliary graph. Solve a min cost flow problem on the augmented original graph to produce the Euler circuit.
Benavent et al.'s Heuristic for the WPP [11]	As a preprocessor to Win's heuristic, identify a set of edges for which $\|c_{ij} - c_{ji}\|$ is large relative to most other edges in the graph. Solve a min cost flow problem, assuming that these edges will be traversed in the cheaper direction. Then, proceed as in Win's heuristic.
Christofides's Heuristic for the DRPP [10]	Solve a min cost spanning arborescence on an auxiliary graph where each required connected component from the original graph is collapsed into a single node. Mark arcs in the arborescence as required, and then solve the DCPP on the resulting graph. Repeat the process, rooting the arborescence in different required connected components while keeping track of the best solution.
Benavent et al.'s Heuristic for the WRPP [11]	Solve a min cost spanning tree on an auxiliary graph where each required connected component from the original graph is collapsed into a single node. Make each edge in the spanning tree solution required, and apply Benavent et al.'s WPP heuristic.

Table 2.3: A summary of the solvers in the library. Solution quality is given as average deviation from optimality presented in the cited references. An asterisk (*) denotes that no computational results were given in the reference.

for the augment-route approach for solving an arc routing problem.

2.5 Problem Setting and Algorithms

In this section, we describe the problems and respective solvers that are implemented in the library. In general, the algorithms proceed by augmenting the underlying network to satisfy the requirements for the existence of an Eulerian circuit (i.e., the augmented graph is Eulerian) in a way that minimizes the aggregate cost of the added elements. An overview of the solution strategies is given in Table 2.3.

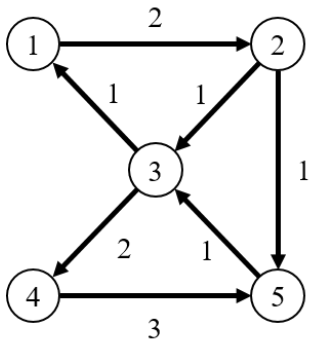


Figure 2.2: An instance of the DCPP. The number inside a node is a vertex id, while the number adjacent to each arc denotes the cost of traversing the corresponding arc. Since this is a directed graph, the arcs in the graph can only be traversed in one direction, from the arrow tail to the arrow head.

2.5.1 Directed Chinese Postman Problem

The DCPP is given by:

$$\text{minimize} \quad \sum_{i \text{ or } j \in \{D^+ \cup D^-\}} c_{ij} x_{ij} \tag{2.1}$$

subject to:

$$\sum_{j \in D^+} x_{ij} = -\delta(i), \forall i \in D^- \tag{2.2}$$

$$\sum_{i \in D^-} x_{ij} = \delta(j), \forall j \in D^+ \tag{2.3}$$

$$x_{ij} \in \mathbb{Z}_+^0 \tag{2.4}$$

The variable x_{ij} is the number of times a shortest path is added from node i to node j in the augmented graph. c_{ij} is the cost of the shortest path from node i to node j. The objective function (2.1) is the total additional cost incurred by the augmentation. Constraints (2.2) and (2.3) ensure that, after we have added these shortest paths, the graph is symmetric.

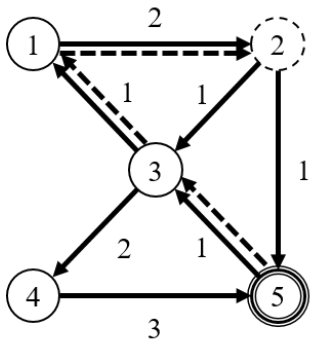


Figure 2.3: The optimal solution to the DCPP in Figure 2.2. Node 5 belongs to D^+ and node 2 belongs to D^- in the initial phase of the algorithm. Arcs added as part of the min cost flow solution are shown as dotted arcs. The optimal Euler circuit is $1-2-5-3-4-5-3-1-2-3-1$. The cost of the solution is the sum of the arc costs $(2(2)+2(1)+1+2+2(1)+1+3=15)$.

2.5.1.1 Exact Algorithm for the Directed Chinese Postman Problem

We solve a min cost flow problem on the underlying directed graph (see Thimbleby [4]) where the supply of vertex i is given by $\delta(v_i)$. A vertex with a negative $\delta(v_i)$ indicates demand. For each unit of flow along an arc in the solution to the resulting flow problem, we add a copy of the arc to the graph. In this way, every vertex in the augmented graph will have $\delta(v)=0$, ensuring that each time a vertex is visited, it is possible to leave it along an arc that has not yet been traversed. It is also possible to show that this a least-cost symmetric augmentation. Figures 2.2 and 2.3 show a small graph and the augmentation produced by the algorithm. The graph in Figure 2.3 is symmetric and, therefore, has an Euler circuit. Pseudocode for this algorithm is given in Table 2.4.

Algorithm 1 DCPP Exact Solver

```
1: procedure DirectedEuler(g)
2:     for vertex v ∈ V do
3:         δ(v) ← in-degree − out-degree
4:         supply(v) ← δ(v)
5:     end for
6:     Solve a min cost flow over G
7:     for i = 1 : |E| do
8:         for j = 1 : flow(e_i) do
9:             Add copy of e_i to G
10:        end for
11:    end for
12: end procedure
```

Table 2.4: Outline of Edmonds and Johnson's [4] exact algorithm for the DCPP.

2.5.2 Undirected Chinese Postman Problem

The UCPP is given by:

$$\text{minimize} \quad \sum_{(i,j)\in E} c_{ij}x_{ij} \tag{2.5}$$

subject to:

$$\sum_{(i,j)\in E_v} (x_{ij}+1) \equiv 0 \ mod\ 2, \forall v \in V \tag{2.6}$$

$$x_{ij} \in \mathbb{Z}_+^0 \tag{2.7}$$

In this formulation, x_{ij} is the number of additional copies of edge (i, j) in our augmented graph. We minimize the added cost, while ensuring every vertex has even degree in the augmented graph (constraints (2.5) and (2.6) achieve this).

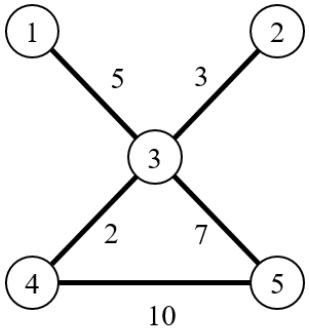


Figure 2.4: An instance of the UCPP. The number inside a node is a vertex id, while the number adjacent to each edge denotes the cost of traversing the edge.

2.5.2.1 Exact Algorithm for the Undirected Chinese Postman Problem

We begin by solving a min cost matching over the odd degree nodes in the original graph where costs are given by shortest path distances. We then add a copy of each edge in the shortest paths between matched vertices. All vertices in this augmentation are even, thereby guaranteeing an Euler circuit. Edmonds and Johnson prove [5] that this is the least-cost way of achieving evenness. In Figures 2.4 and 2.5, we show this process. Pseudocode for this algorithm is given in Table 2.5.

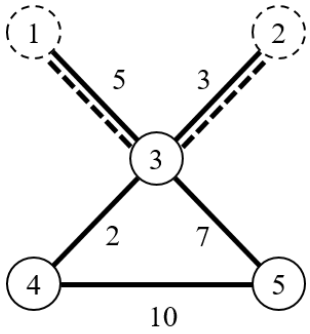


Figure 2.5: The optimal solution to the UCPP in Figure 2.4. Nodes with dotted borders (1 and 2, in this case) are identified as odd degree nodes in the initial phase of the algorithm. Edges that were added as part of the matching solution are shown as dotted lines.

Algorithm 2 UCPP Exact Solver

```
1:  procedure UndirectedEuler(g)
2:      V_matching ← V_odd
3:      E_matching ← ∅
4:      for vertex i ∈ V_odd do
5:          for vertex j ∈ V_odd do
6:              Add e_ij to E_matching with c_ij = sp_ij
7:          end for
8:      end for
9:      Let G_matching = (V_matching, E_matching)
10:     Solve a min cost matching over G_matching
11:     for e_ij ∈ matching do
12:         Add copy of e_ij to G
13:     end for
14:     Return Hierholzers(G)
15: end procedure
```

Table 2.5: Outline of Edmonds and Johnson's [5] exact algorithm for the UCPP

2.5.3 Mixed Chinese Postman Problem

The MCPP is given by:

$$\text{minimize} \quad \sum_{s \in \{A \cup \hat{E} \cup \check{E}\}} c_s x_s \tag{2.8}$$

subject to:

$$y'_e + y'_{\tilde{e}} \geq 1, \forall e \in E \tag{2.9}$$

$$x_s = y'_s + y_s, \forall s \in A \cup \hat{E} \cup \check{E} \tag{2.10}$$

$$\sum_{s \in \delta_v^+} x_s - \sum_{s \in \delta_v^-} x_s = 0, \ \forall v \in V \tag{2.11}$$

$$y'_a = 1, \ \forall a \in A \tag{2.12}$$

$$y'_e \in \{0, 1\}, \ \forall e \in \hat{E} \cup \check{E} \tag{2.13}$$

$$y_s \in \mathbb{Z}_+^0 \tag{2.14}$$

The objective function given by (2.8) is the cost of the tour. y'_s is 1 if link s is traversed, and 0 if it is not traversed. y_s is the number of additional times link s is traversed. The set $\hat{E}$ contains edges that are traversed from i to j in the solution, while the set $\check{E}$ contains edges that are traversed from j to i. Similarly, the subscripts e and $\tilde{e}$ correspond to traversing edge e forward (from i to j) and backward (from j to i), respectively. δ_v^+ and δ_v^- denote the set of edges and arcs which start at, or end at, vertex v respectively. Therefore, x_s is the total number of times link s is traversed. Constraint (2.9) ensures that each edge is traversed at least once. Constraint (2.10) defines x_s. Constraint (2.11) ensures symmetry. Constraint

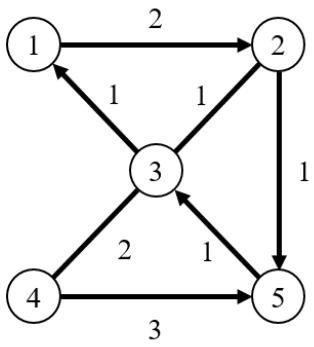


Figure 2.6: An instance of the MCPP. The number inside the node is a vertex id, while the number adjacent to each link denotes the cost of traversing the corresponding link. Arrows denote arcs that can only be traversed from tail to head, while lines without arrows denote edges.

(2.12) ensures that arcs are traversed at least once. Constraint (2.13) is the binary constraint for y'_s. Constraint (2.14) is the integrality constraint for the y_s.

2.5.3.1 The Even-Symmetric-Even Heuristic

We augment the graph to produce an Eulerian graph in the hope of finding an Eulerian circuit. Recall that in order for a mixed graph to be Eulerian, it must be balanced (i.e., for each subset of nodes S, $|\{e_{ij} \in E : i \in V \text{ and } j \in V \setminus S\}| \geq (|\{a_{ij} \in A : i \in V \text{ and } j \in V \setminus S\}| - |\{a_{ij} \in A : i \in V \setminus S \text{ and } j \in V\}|)$. Intuitively, this condition ensures that there are enough (undirected) edges to account for the difference between the number of one-way streets in and out of a portion of the graph.

Checking whether a mixed graph is balanced is computationally intractable because the number of subsets $|S|$ is exponential in the number of vertices $|V|$. Therefore, the heuristic relies on the fact that it is sufficient but not necessary for a

balanced graph to be even and symmetric (see Frederickson [6]).

The Even-Symmetric-Even heuristic has three phrases. In the first phase, it achieves evenness by carrying out a min cost matching among the odd-vertices. In the second phase, it achieves symmetry by using a min cost flow algorithm on the asymmetric nodes. In the third phase, it restores evenness by looking for circuits that may be eliminated while preserving the symmetry achieved in the second phase. This process is shown in Figures 2.6 and 2.7. Pseudocode for this heuristic is given in Table 2.4.

1. *Phase I, Even.* Solve the UCPP on the original graph by treating all arcs as edges. This produces an augmented graph G^E.

2. *Phase II, Symmetric.* Solve a min cost flow problem on G^E by treating each edge (u, v) as four arcs (the first two (u, v) and (v, u) with cost equal to the original edge cost and infinite flow capacity, and two (u, v) and (v, u) with zero cost, and flow capacity of 1). In the flow solution, if arc (u, v) is traversed only once, or arc (v, u) is traversed only once, then we orient the edge in that direction; otherwise edge (u, v) remains as an edge in our output graph G^S.

3. *Phase III, Even*: Using a greedy procedure, search for circuits that have paths between any odd-degree nodes left in G^S. If there are none, Phase III is skipped. These paths must alternate between using arcs or oriented edges added in Phase II, and using edges left undirected by Phase II. For example, if there were four odd-degree nodes remaining (v_1, v_2, v_3, v_4), an eligible circuit would be a path from v_1 to v_2 of added arcs, a path from v_2 to v_3 of unoriented

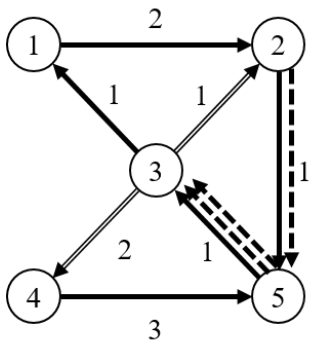


Figure 2.7: A solution to the MCPP instance in Figure 2.6 procduced by Frederickson's algorithm [6]. The dotted arc $(2, 5)$ is added in the initial Even phase, while all other dotted arcs are added in the Symmetric phase. The double arrows (3,4) and (3,2) indicate that the edges in the original graph were oriented in the corresponding direction.

Algorithm 3 MCPP Heuristic Solver

```
1: procedure MixedFrederickson(g)
2:     G_even ← EvenDegree(G)
3:     G_symm ← Symmetric(G_even)
4:     G_final ← EvenParity(G_symm)
5:     Return Hierholzers(G_final)
6: end procedure
```

Table 2.6: Outline of Frederickson's heuristic [6] for the MCPP

edges, a path from v_3 to v_4 of added arcs, and a path from v_4 to v_1 of unoriented edges. This ensures that only the parity of the odd-degree nodes is changed, while assigning a direction to all remaining undirected edges. After we find one of these alternating paths, we orient it (either direction will be equivalent) and duplicate arcs and oriented edges along the path that follow the orientation, while deleting arcs that are in the opposite direction. For the segments of the circuit that have edges, we orient them in the direction we have chosen to orient the circuit.

2.5.3.2 Shortest Additional Path Heuristic

The initial step of the Shortest Additional Path Heuristic (SAPH, [7]) is identical to the second phase of the Even-Symmetric-Even heuristic where the graph is transformed into a symmetric one. After this phase is completed, links in the graph are characterized as belonging to one of six categories shown in Figure 2.8. These categories are based on how many copies of a given link have been added in the augmentation in the first phase, and, in the case of an edge, which direction it has been assigned. The remaining phases of the algorithm attempt to delete some of these added links by adding and subtracting paths in the graph.

This second phase begins by exploiting two ideas. First, suppose that an edge or arc was added to the original graph, and oriented from node A to node B. If the shortest path cost from node A to B is less than the cost of traversing this added link, then we replace the link with the shortest path from A to B (see Figure 2.9). We call this shortest path replacement. Pseudocode for this procedure is given in Table 2.7. Second, if an edge was oriented from node A to B, and the two shortest paths have costs that sum less than zero, then it is advantageous to traverse the shortest path from A to B, service the edge in the opposite direction (from B to A), and then traverse the second shortest path from A to B. Although this second case may seem unusual because the shortest path costs will generally be positive, it arises when we consider deleting added arcs and incurring savings equal to its cost. This is also the reason why the two shortest paths may not be the same. For example, in Figure 2.10, two paths from v_1 to v_2 are added. One of these paths

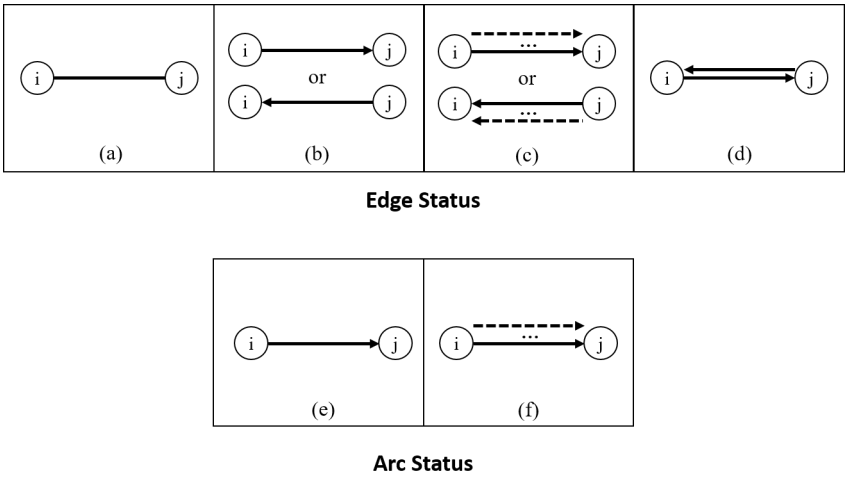


Figure 2.8: Edges and arcs in G must end up in one of the following configurations in G^*. If an edge remains undirected, it is type a. If an edge gets directed, but not copied, it is type b. If an edge gets directed and copied, but all copies are in the same direction, then it is type c. If an edge gets copied once, and oriented in the opposite direction as the original, it is type d. If an arc is not copied, it is type e. If an arc is copied, it is type f. The ellipses (...) in the diagrams for type c and type f indicate that there may be many copies.

costs -7. Suppose that an arc of cost 9 from v_4 to v_3 was added in the first phase of the heuristic and that $c_{13} = c_{42} = 1$. Then, the path $v_1 - v_3 - v_4 - v_2$ would have cost $1 - 9 + 1 = -7$. We call this procedure reversal. Pseudocode for this procedure is given in Table 2.8.

SAPH is described below.

1. Given a mixed graph G, generate a graph $G^* = (N, M, U)$ and a set of added arcs M^* by solving Phase II of Even-Symmetric-Even on G. Generate a graph $G_M = (N, E + E_M, A + A_M)$ by solving Phase I of Even-Symmetric-Even on G, where E_M and A_M are the sets of edges and arcs added from the matching.

2. Choose a random edge or arc in G^* of type a, c, d or f.

3. Initialize two graphs $G^1_{ij} = G$ and $G^2_{ij} = G$.

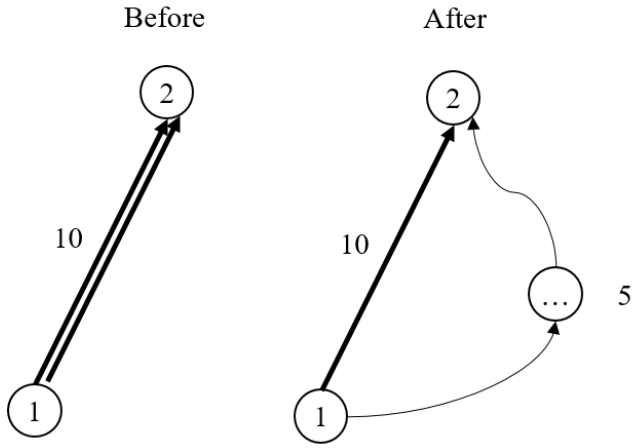


Figure 2.9: Illustration of shortest path replacement where we replace an added arc with a cheaper shortest path. The node with the ellipsis (...) denotes the shortest path from vertex 1 to vertex 2. Originally (in the before panel), we have added a copy of the arc from vertex 1 to vertex 2 in an earlier phase of the algorithm. It is replaced in the after panel by a shortest path from vertex 1 to vertex 2, which costs 5 instead of 10, producing a savings of 5.

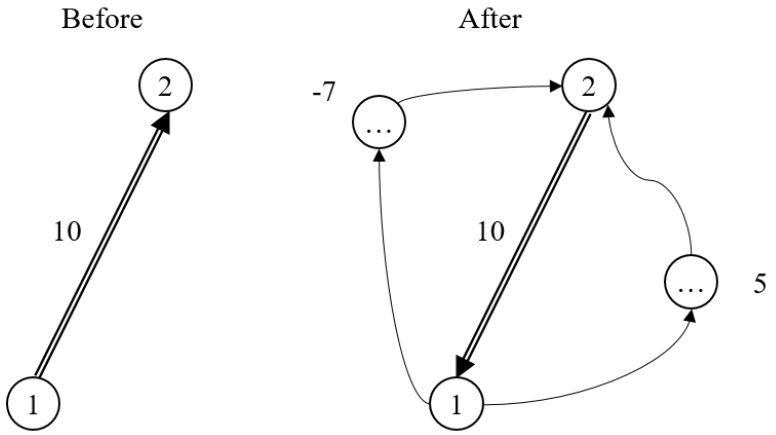


Figure 2.10: Illustration of reversal where we reverse the orientation of an edge and add two paths from node i to j which sum to a negative cost. Originally (in the before panel), an edge between vertex 1 and vertex 2 was oriented from vertex 1 to vertex 2 in an earlier phase of the algorithm. The assigned orientation is reversed (from vertex 2 to vertex 1) and the two shortest paths from vertex 1 to vertex 2 are added. In this case, their distances sum to -2, producing a savings of 2. Intuitively, when a postman arrives at vertex 1, the postman would previously proceed to vertex 2. Now, in the after panel, the postman traverses the first shortest path to vertex 2, returns to vertex 1 via the edge, and then traverses the second shortest path, ending at vertex 2.

4. Perform *Cost modification 1* on G_{ij}^1.

5. Perform *Cost modification 2* on G_{ij}^1 and G_{ij}^2.

6. Apply the first shortest paths idea to the selected edge or arc.

7. Repeat all steps until there are no more edges of type a, c, d or f.

8. Select a random edge of type b.

9. Apply the second shortest paths idea to the selected edge or arc.

10. Go back to Step 8 until there are no more edges of type b.

11. If we applied the second shortest paths idea to make any improvements, go back to Step 1.

12. If there are any edges (i, j) of type a left in G^*, orient them from i to j, and add a copy (j, i) oriented in the opposite direction.

We now specify the cost modification procedures.

Cost modification 1: This procedure tries to force our shortest paths algorithm to traverse links from the matching solution.

1. Given G_{ij}, G_M, and a nonpositive number K, find all edges (f, g) in G_{ij} that are also in E_M. In G_{ij}, set their costs $c_{fg} = c_{gf} = K$.

2. Locate in G_{ij} all arcs from A_M. If the arc is type f in G^*, then set the costs $c_{fg} = c_{gf} = K$. If the arc is type e, then set $c_{fg} = 0, c_{gf} = \infty$.

Algorithm 4 SAPH Concept 1

procedure COST MODIFICATION 1(An added arc $a_{ij} \in G$)
    $c_{ij} \leftarrow \infty$
    Cost modify G
    Calculate shortest path from i to j, with length sp_{ij}
    if $sp_{ij} < c_{ij}^{orig}$ **then**
        Delete a copy of a_{ij} in G
        Add a copy of sp_{ij} to G
    end if
end procedure

Table 2.7: Outline of the shortest path replacement improvement procedure.

Algorithm 5 SAPH Concept 2

procedure COST MODIFICATION 2(A oriented edge $e_{ij} \in G$)
    $c_{ij} \leftarrow \infty$
    Cost modify G
    Calculate two shortest paths from i to j, with lengths sp_{ij}^1 and sp_{ij}^2
    if $sp_{ij}^1 + sp_{ij}^2 < c_{ij}$ **then**
        Change the orientation of e_{ij} in G
        Add a copy of sp_{ij}^1 and sp_{ij}^2 to G
    end if
end procedure

Table 2.8: Outline of the reversal improvement procedure.

Cost modification 2: This procedure tries to force our shortest paths algorithm to traverse links that will benefit from our two improvement procedures at the same time as we examine our selected link which may get deleted as part of a shortest path from i to j.

1. Given graphs G_{ij} and G^*, find all edges (f, g) in G^* that are type a or d. Let c_{fg}^* denote the cost of link (f, g) in the original graph G. Then, set the costs c_{fg} and c_{gf} in G_{ij} to be $-c_{fg}^*$ and $-c_{gf}^*$.

2. In G^*, find all links (f, g) of type c or f. Set the cost c_{gf} in G_{ij} to $-c^*_{fg}$.

3. At whatever point in the process this procedure is being called, set the cost of the selected link in G_{ij} to ∞ in both directions, that is, $c_{fg} = c_{gf} = \infty$.

2.5.4 Windy Postman Problem

The WPP is given by:

$$\text{minimize} \quad \sum_{e^+ \in E^+} c_{e^+} x_{e^+} + \sum_{e^- \in E^-} c_{e^-} x_{e^-} \tag{2.15}$$

subject to:

$$\sum_{e^+ \in E^+} x_{e^+} - \sum_{e^- \in E^-} x_{e^-} = 0, \forall v \in V \tag{2.16}$$

$$x_{e^+} + x_{e^-} \geq 1, \forall e \in E \tag{2.17}$$

$$x_{e^+}, x_{e^-} \in \mathbb{Z}^0_+, \forall e \in E \tag{2.18}$$

The formulation of the CPP on a windy graph is similar to the formulation of the CPP on an undirected graph. In this formulation, x_{e^+} and x_{e^-} are the number of times an edge e is traversed in the forward and reverse direction, respectively. Constraint (2.16) enforces symmetry for each vertex, while constraints (2.17) and (2.18) are the traversal and integrality requirements.

2.5.4.1 Win's Algorithm

With the WPP, the solution strategy is different from the strategies for the UCPP, DCPP, and MCPP. Previously, we could calculate the cost of an augmen-

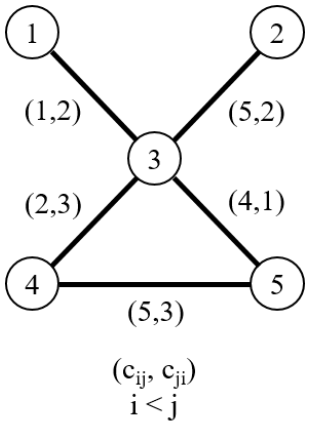


Figure 2.11: An instance of the WPP. Each edge has two costs denoted by c_{ij} and c_{ji} where c_{ij} is the cost of traversing the edge from i to j where $i < j$ and c_{ji} is the cost of traversing the edge from j to i.

tation ahead of time. We cannot do this for the WPP because we do not know which direction the postman will traverse the edges in the circuit. The exact cost of adding an edge is more difficult to calculate.

Win's algorithm [8] addresses this difficulty by considering average costs. It solves the UCPP on the graph $G_{\bar{E}}$ with costs $\bar{c}_{ij} = \min(sp_{ij} + sp_{ji})$ (see Figures 2.11 and 2.12). This produces an Eulerian augmentation to the original graph. We then use a polynomial time algorithm that generates the optimal tour on this augmented graph. The full procedure is given in Table 2.9.

1. Given the Eulerian graph G, form the directed graph $D_G = (V, A)$ where the vertex set is identical to the vertex set of G. For each edge in G, if $c_{ij} < c_{ji}$, then arc (i, j) is added to A. Otherwise, arc (j, i) is added to A.

2. Create a second directed graph $D' = (V, A')$ by, for each arc $(i, j) \in A$, adding three arcs to A': one arc (i, j) with cost c_{ij} and infinite capacity, one arc (j, i)

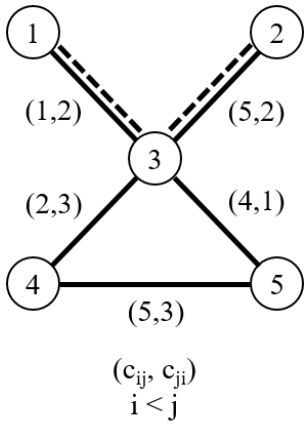


Figure 2.12: A solution to the WPP from Figure 2.11. The dotted edges are added as part of the flow solution.

with cost c_{ji} and infinite capacity, and one arc $(j,i)'$ with cost $\dfrac{c_{ji}-c_{ij}}{2}$ and capacity two. The third arc is an artificial arc.

3. Solve a min cost flow problem on D', with demands calculated in the same way as in the DCPP on D_G.

4. Construct an Eulerian directed graph $D'' = (V, A'')$ in the following way. If, in the flow solution, there is 0 flow along the arc $(j,i)'$, then add $1 + x_{ij}$ copies of arc (i,j) to A''. Otherwise, add $1 + x_{ji}$ copies of arc (j,i) to A''. The Euler circuit on this directed graph is an optimal solution to the WPP on G.

2.5.4.2 Algorithm of Benavent et al.

The second algorithm for the WPP from Benavent et al. [11] is an improvement over Win's original algorithm. It anticipates the results of the min cost flow problem that produces the optimal windy tour. Edge costs are modified before the matching

Algorithm 6 WPP Heuristic Solver

procedure WIN'S ALGORITHM(g)
  $V_{matching} \leftarrow V$
  $E_{matching} \leftarrow \emptyset$
  for $v_i \in V_{odd}$ **do**
    for $v_j \in V_{odd}$ **do**
      Add e_{ij} to $E_{matching}$ with $c_{ij}^{matching} = \min\left(sp_{ij}^{avg}, sp_{ji}^{avg}\right)$
    end for
  end for
  $G_{matching} = (V_{matching}, E_{matching})$
  Solve a min cost matching on $G_{matching}$
  Add a copy of each edge in the matching to G
  $V_{flow} \leftarrow V$
  $A_{flow} \leftarrow \emptyset$
  for Windy edge $w_{ij} \in E$ **do**
    Let $c_{ij}^{orig} < c_{ji}^{orig}$
    Add a_{ij} to A_{flow} with cost c_{ij}
    Add a_{ji} to A_{flow} with cost c_{ji}
    Add an artificial a_{ji} to A_{flow} with cost $\frac{c_{ji} - c_{ij}}{2}$ and flow capacity 2
  end for
  $G_{flow} = (V_{flow}, A_{flow})$
  Solve a min cost flow on G_{flow}
  $V_{ans} \leftarrow V$
  $A_{ans} \leftarrow \emptyset$
  for Windy edge $w_{ij} \in E$ **do**
    if flow along $a_{ij} = 0$ **then**
      Add $x + 1$ copies of a_{ij} to A_{ans}
    else
      Add $x + 1$ copies of a_{ji} to A_{ans}
    end if
  end for
  $G_{ans} = (V_{ans}, A_{ans})$
  Return Hierholzers(G_{ans})
end procedure

Table 2.9: Outline of Win's heuristic [8] for the WPP.

is solved to produce an Eulerian undirected graph. Pseudocode for this algorithm is given in Table 2.10.

1. Given the original windy graph $G = (V, E)$, calculate the average edge cost for the entire graph ($C_a = \frac{1}{2|E|}\sum_{(i,j)\in E} c_{ij} + c_{ji}$). Now, consider edge set $E_1 = (i, j) \in E : \{|c_{ij} - c_{ji}|\} > K(C_a)$. Let $E_2 = E \backslash E_1$.

2. Construct a directed graph $G_R^d = (V, A')$ where, for each $e \in E$, add two arcs in A', (i, j) with cost c_{ij} and infinite capacity, and (j, i) with cost c_{ji} and infinite capacity. For each $e \in E_1$, add an additional artificial arc (j, i) with cost $\frac{c_{ji} - c_{ij}}{2}$ and a capacity of two.

3. Solve a min cost flow problem with demands given by a reduced graph $G' = (V, A)$. The reduced graph contains an arc (i, j) for each edge $(i, j) \in E_1$. We assume $c_{ij} < c_{ji}$ so that the arcs in A are in a cheaper direction.

4. Compile a list L of edges such that $e \in E_1$ and, in the flow solution, there is positive flow across the corresponding (non-artificial) arcs. Also, $e \in E_2$ and, in the flow solution, there is at least a flow of two across its corresponding (non-artificial) arcs.

5. For each edge $e \in L$, set the cost to 0 in the original graph. Compute the min cost matching, as in Win's algorithm. Restore costs to the costs in the original graph. Proceed as in Win's algorithm.

2.5.5 Directed Rural Postman Problem

The DRPP is given by:

$$\text{minimize} \qquad \sum_{a \in A} c_a x_a \tag{2.19}$$

subject to:

$$x_a \geq 1, \ \forall a \in A_R \tag{2.20}$$

$$\sum_{\{a \in A : he_a = i\}} x_a - \sum_{\{a \in A : ta_a = i\}} x_a = 0 \ , \ \forall i \in V \tag{2.21}$$

$$\sum_{\{a \in A : ta_a \in S \not\ni he_a\}} x_a \geq 1, \forall \emptyset \neq S \subset V, |S| \leq \lfloor \frac{|V|}{2} \rfloor \tag{2.22}$$

$$x_a \in \mathbb{Z}_+^0 \tag{2.23}$$

The objective function (2.19) is the cost of the tour. Constraint (2.20) enforces traversal of required arcs. Constraint (2.21) enforces the path to be a circuit. Constraint (2.22) eliminates subtours. Constraint (2.23) imposes integrality.

2.5.5.1 Christofides's Algorithm

Christofides's algorithm [23] simplifies the original graph by discarding the unrequired nodes and arcs and connecting the required connected components of the graph. A min cost flow problem is solved over the remaining graph to obtain a feasible solution to the DRPP.

1. Given the input graph $G = (V, A_R \cup A_{NR})$, define the vertex set V_R to be the

set of nodes that have at least one required arc incident. Consider the graph $G_R = (V_R, A_R)$. We form a complete graph $G' = (V_R, A_R \cup A_S)$ by connecting all vertices in V_R with arcs (i, j) that have cost equal to the shortest path in G between node i and node j. These costs are finite because the graph is strongly connected. The added arcs are in the set A_S. Remove from G' any arc $(i, j) \in A_S$ that has cost $c_{ij} = c_{ik} + c_{kj}$ for some $k \in V_R$ and is a duplicate of an arc in A_R.

2. Starting with the directed graph G', collapse each connected required component into a node. Solve the shortest spanning arborescence (SSA) problem on this collapsed graph. Add arcs found in the SSA to a set T_{t_a} to indicate that the SSA was rooted in the connected component t_a.

3. Solve a min cost flow on the graph G' with demands calculated as out-degree minus in-degree relative to the arc set $A_R \cup T_{t_a}$ where every arc has infinite capacity. Let f_{ij} be the amount of flow through arc (i, j) in the flow solution. Add f_{ij} copies of arc (i, j) to an arc set F. The final feasible solution graph is given by $G_S = (V_R, A_R \cup T_{t_a} \cup F)$.

We repeat the algorithm with k different SSAs where k is the number of required components of the simplified graph G'. The SSA requires a choice of root node. We solve using each node in the collapsed graph as the root, and choose the best solution. This process is depicted in Figures 2.13 and 2.14. Pseudocode for this algorithm is given in Table 2.11.

Algorithm 7 WPP Heuristic Solver

procedure BENAVENT ET AL.'S H1 HEURISTIC(g)
  $C_a \leftarrow$ avg. cost of traversal in G
  $E_1 \leftarrow \emptyset$
  $E_2 \leftarrow \emptyset$
  for Windy edge $w_{ij} \in E$ **do**
    if $|c_{ij} - c_{ji}| > K * C_a$ **then**
      Add w to E_1
    else
      Add w to E_2
    end if
  end for
  $A' = \emptyset$
  for Windy edge $w_{ij} \in E$ **do**
    Let $c_{ij}^{orig} < c_{ji}^{orig}$
    Add a_{ij} to A_{flow} with cost c_{ij}
    Add a_{ji} to A_{flow} with cost c_{ji}
    if $w_{ij} \in E_1$ **then**
      Add an artificial a_{ji} to A_{flow} with cost $\frac{c_{ji} - c_{ij}}{2}$ and flow capacity 2
    end if
  end for
  $G_R^d = (V, A')$
  Solve a min cost flow problem on G_R^d
  $L \leftarrow \emptyset$
  for Windy edge $w_{ij} \in E$ **do**
    if $w_{ij} \in E_1$ and flow(a_{ij}) + flow(a_{ji}) > 0 **then**
      Add w_{ij} to L
    end if
    if $w_{ij} \in E_2$ and flow(a_{ij}) + flow(a_{ji}) > 1 **then**
      Add w_{ij} to L
    end if
  end for
  for Windy edge $w_{ij} \in L$ **do**
    Set $c_{ij} = c_{ji} = 0$
  end for
  Perform avg. min cost matching over G
  Add a copy of each edge included in the matching
  Reset all costs back to original
  Construct the optimal windy circuit on G
end procedure

Table 2.10: Outline of the heuristic by Benavent et al. [11] for the WPP.

Algorithm 8 DRPP Heuristic Solver

procedure ChristofidesDRPPSolver(g, A_R)
  $G_R \leftarrow (V, A_R)$
  $C = C_1, C_2, ...$ =connected components of G_R
  $V_{Arb} \leftarrow \emptyset$
  $A_{Arb} \leftarrow \emptyset$
  for Component $c_i \in C$ **do**
    Add v_i to V_{Arb}
  end for
  for Arc $a_{ij} \in A$ **do**
    if $v_i \in C_i$ and $v_j \in C_j$ and $i \neq j$ **then**
      Add a_{ij} to A_{Arb}
    end if
  end for
  $G_{Arb} = (V_{Arb}, A_{Arb})$
  Solve a Minimum Spanning Arborescence over G_{Arb}
  for Arc $a \in$ MSA **do**
    Set a to required in G
  end for
  Solve a DCPP over G where supplies and demands given by G_R
  Return Hierholzers(G)
end procedure

Table 2.11: Outline of Christofides's heuristic [23] for the DRPP.

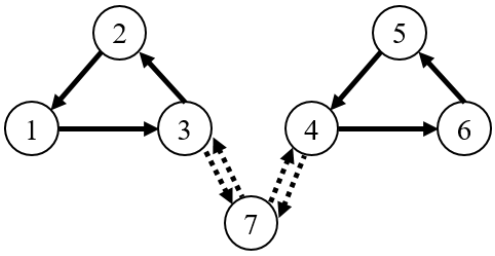


Figure 2.13: An instance of the DRPP. Solid arcs are required (our solution must traverse them at least once), while dotted arcs are not.

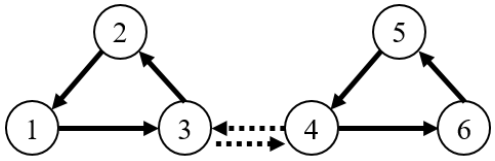


Figure 2.14: The solution to the DRPP on the graph from Figure 2.13. Node 7 has been deleted because of the graph simplification. However, c_{34} will be increased to $c_{37} + c_{74}$, and the same change will be made to c_{43}.

2.5.6 Windy Rural Postman Problem

The WRPP is given by:

$$\text{minimize} \qquad \sum_{e^+ \in E^+} c_{e^+} x_{e^+} + \sum_{e^- \in E^-} c_{e^-} x_{e^-} \tag{2.24}$$

subject to:

$$\sum_{e^+ \in E^+} x_{e^+} - \sum_{e^- \in E^-} x_{e^-} = 0, \forall v \in V \tag{2.25}$$

$$x_{e^+} + x_{e^-} \geq 1, \forall e \in E_R \tag{2.26}$$

$$\sum_{i \in S, j \in V \setminus S} x_{ij} \geq 1, \forall S \text{ required cut-sets} \tag{2.27}$$

$$x_{e^+}, x_{e^-} \in \mathbb{Z}^0_+, \forall e \in E \tag{2.28}$$

The formulation of the WRPP is similar to the formulation of the WPP. The

only difference is that constraint (2.26) is enforced for the required edges and not on the entire edge set. The variables x_{e^+} and x_{e^-} are the number of times an edge e is traversed in the forward and reverse direction, respectively. The objective function given by (2.24) is the cost of the tour. Constraint (2.27) eliminates subtours, where required cut-sets are edge cut-sets between the required connected components of the graph. Constraint (2.25) enforces symmetry for each vertex, while constraints (2.26) and (2.28) are the traversal and integrality requirements.

2.5.6.1 Algorithms from Benavent et al.

The procedures from Benavent et al. [11] are identical to their counterparts for solving the WPP. WRPP1 corresponds to Win's algorithm except that a minimum spanning tree problem must be solved in order to connect the required components of the graph. Pseudocode for this algorithm is given in Table 2.12. The procedure for this follows.

1. Compute the connected components $C_1, C_2, C_3, ...,$ of the graph G_R,
2. Construct the graph G_C where the vertex set V_C contains one vertex for each connected component in G_R.
3. Complete G_C by adding edges e_{ij} with costs $c_{ij} = \min(c_{avg}(sp_{ij}), c_{avg}(sp_{ji}))$.
4. Solve the minimum spanning tree (MST) problem on G_C.
5. If e_{ij} was included in the MST, then set each edge in the shortest path represented by e_{ij} to be required.

Algorithm 9 WRPP Heuristic Solver

procedure BENAVENT'S H1(g, E_R)
  $G_R \leftarrow (V, E_R)$
  $C = C_1, C_2, ...$ =connected components of G_R
  $V_{MST} \leftarrow \emptyset$
  $E_{MST} \leftarrow \emptyset$
  for Component $c_i \in C$ **do**
    Add v_i to V_{MST}
  end for
  for Windy edge $e_{ij} \in E$ **do**
    if $v_i \in C_i$ and $v_j \in C_j$ and $i \neq j$ **then**
      Add e_{ij} to E_{MST} with $c_{ij} = \min(c_{avg}(sp_{ij}), c_{avg}(sp_{ji})$
    end if
  end for
  $G_{MST} = (V_{MST}, A_{MST})$
  Solve a Minimum Spanning Tree over G_{MST}
  for Windy edge $e \in$ MST **do**
    Set e to required in G
  end for
  Solve a WPP over G where degree is determined in G_R
  Return Hierholzers(G)
end procedure

Table 2.12: Outline of Benavent et al.'s H1 heuristic [11] for the WRPP.

2.6 Results

We present runtimes for the solvers in the library. All tests were performed on a MacBook Air (August 2012) with an i5-3427u processor. We use publicly available test instances modeled on real street networks that are posted at http://www.uv.es/corberan/instancias.htm. These instances are grouped into sets of 24 with each set having a fixed $|V| \in \{500, 1000, 1500, 2000, 3000\}$ for a total of 120 instances and varying $|E|$ up to 9085. Our library contains a parser for the format that outputs a graph object that is used as input to our solvers.

Figures 2.15-2.18 give the runtimes in milliseconds (ms) for each of the solvers in the library. Figure 2.15 shows results for the UCPP solver. Figure 2.16 shows results for the DCPP solver. Figure 2.17 compares the runtimes for the two MCPP solvers. Yaoyuenyong's heuristic is shown in red and it is clear that the improved objective values require large runtimes. For the largest instances, the runtimes were less than 20 minutes. Frederickson's were less than two minutes. Figure 2.18 shows the runtimes for the WPP solver. Runtime was increased by trying to modify the solution in anticipation of the optimal Euler circuit calculation. The jump in the runtimes at about 5000 links is due to the set of test instances containing two groups. The set of smaller instances reaches a maximum of 6000 links, while the set of larger instances ranges from 5000 to 9000 links.

For the UCPP and DCPP, we use a graph generator that randomly produces a graph with a specified density, number of vertices, and connectedness (Boolean) as inputs. Most of the work involves producing the solution to the flow or matching problem induced by the original graph. In order to solve the min cost matching problem, we use the publicly available, efficient C++ implementation of the Blossom algorithm presented by Kolmogorov [19]. To call this code from Java, we use a simple function wrapper, with the Java Native Interface, to communicate cross-platform. This may explain why the UCPP solver's performance on smaller problem instances does not monotonically increase with problem size. For small instances, the overhead of calling the function rather than the function itself dominates the runtime.

To validate the quality of the results, we replicated the objective function values in each paper that introduced the methods using the same benchmark instances.

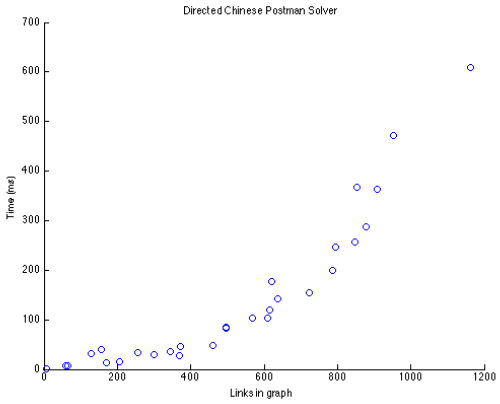


Figure 2.15: Runtimes (ms) for our implementation of the DCPP exact solver to generate the optimal solution.

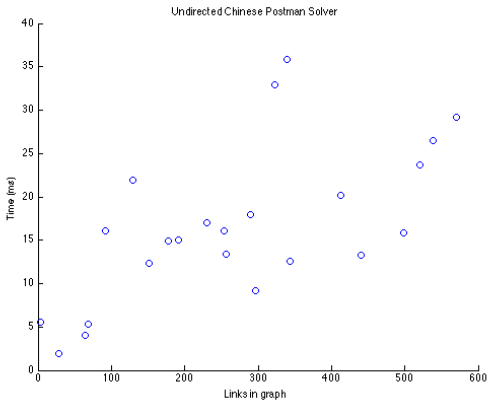


Figure 2.16: Runtimes (ms) for our implementation of the UCPP exact solver to generate the optimal solution.

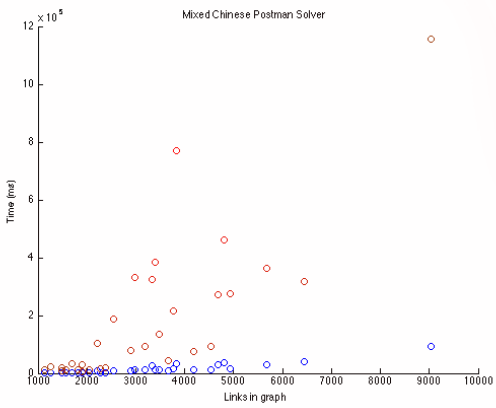


Figure 2.17: Runtimes (ms) for our implementation of Frederickson's algorithm [6] in blue and Yaoyuenyong's algorithm [7] in red for the MCPP instances.

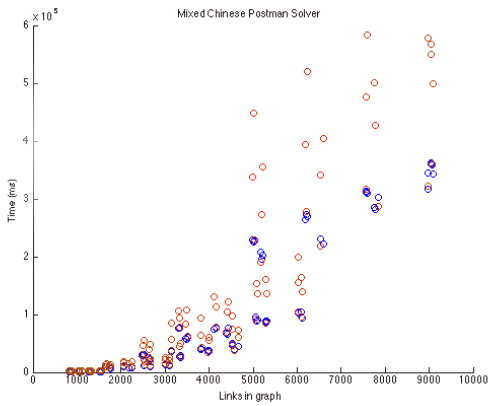


Figure 2.18: Runtimes (ms) for our implementation of Win's algorithm [8] in blue and Benavent et al.'s algorithm [11] in red for the WPP instances.

For the DCPP and UCPP, the papers did not report computational results. However, because these are exact procedures, we compare the results of our solver to the objective function values obtained by the IP formulation given earlier (equations 2.1-2.4, and 2.5-2.7 for the DCPP and UCPP, respectively). The formulations were solved using Gurobi. For the MCPP, Yaoyuenyong [7] presented a set of benchmark instances and computational results for Frederickson's heuristic and the SAPH. We replicated these results. For the WPP and WRPP, Benavent et al. [11] did not present results on individual instances. They instead reported deviation from the optimal solution averaged over the set of Albaida-Madrigueras test instances [24]. We reproduced this average deviation over the same set of test instances. Benavent et al. [11] described an extension of Win's heuristic for the WPP that applies to the WRPP. For instances where all edges are required, the two heuristics are the same. Therefore, we use the averages reported by Benavent et al. for this procedure to validate Win's heuristic. For the DRPP, we replicated the results presented by Campos and Savall [10] that gives a comparison of several DRPP heuristics.

We also include a suite of tests using the JUnit framework [25] to provide coverage of most non-trivial functions. JUnit is one of several testing frameworks that provides Java coders the ability to write a set of test routines. Tests contain several assert statements that compare the output of the test with an expected value. The test is said to fail if the assertion is false. For the code covered by the tests (logic paths executed by the test subroutines), running these tests whenever the code is modified allows the coder to isolate changes that caused the tests to fail. Many popular integrated development environments have plug-ins that allow for tests written using JUnit to be run for continuous integration, automatically alerting the coder as soon as the violating change is made without a manual run of the tests. These tests give potential contributers an easy way to validate changes without having to be familiar with parts of the code that they did not write. Users of the library may also examine these tests as additional documentation of the intended use of each function.

2.7 Conclusions

We implemented a suite of solvers for well-known arc routing problems and presented computational results on several sets of benchmark instances. For the UCPP and DCPP, we provided exact solvers. For the MCPP, WPP, DRPP, and WRPP, we provided heuristic solvers. We implemented and discussed the underlying code architecture. We demonstrated its flexibility in accommodating a variety of graph types and problem features. The code is released under the MIT License

and is available at github.com/olibear/ArcRoutingLibrary. We plan to extend the architecture in future work, including problems with alternative objective functions and multiple vehicles.

Chapter 3: An Open Source Desktop Application for Generating Arc Routing Benchmark Instances

3.1 Introduction

Arc routing problems date back to the well-known Königsberg bridges problem [26, 27]. Both from a theoretical and a practical point of view, research interest in this area has increased dramatically over the last decade or so. This is, perhaps, best evidenced by the comprehensive 2014 volume by Corberán and Laporte [28], the detailed annotated bibliographies of 2010 and 2017 [29, 30], and the two recent, international conferences dedicated to the study of arc routing problems held in Copenhagen (2013) and Lisbon (2016). In Figure 3.1, we see how the number of arc routing publications has grown since 2000.

Optimization algorithms and heuristic procedures for arc routing problems often use benchmark instances to validate and demonstrate performance [31–35]. Ideally, these benchmark instances try to capture the features of real-world street networks encountered in practice. Typically, benchmark instances are artificially generated and only approximate real-world networks [36]. In Figure 3.2, we show the two types of networks used in the literature: a real-world, GIS-based network and a random network.

Open street maps (OSM) is an open-source, user-driven map database that

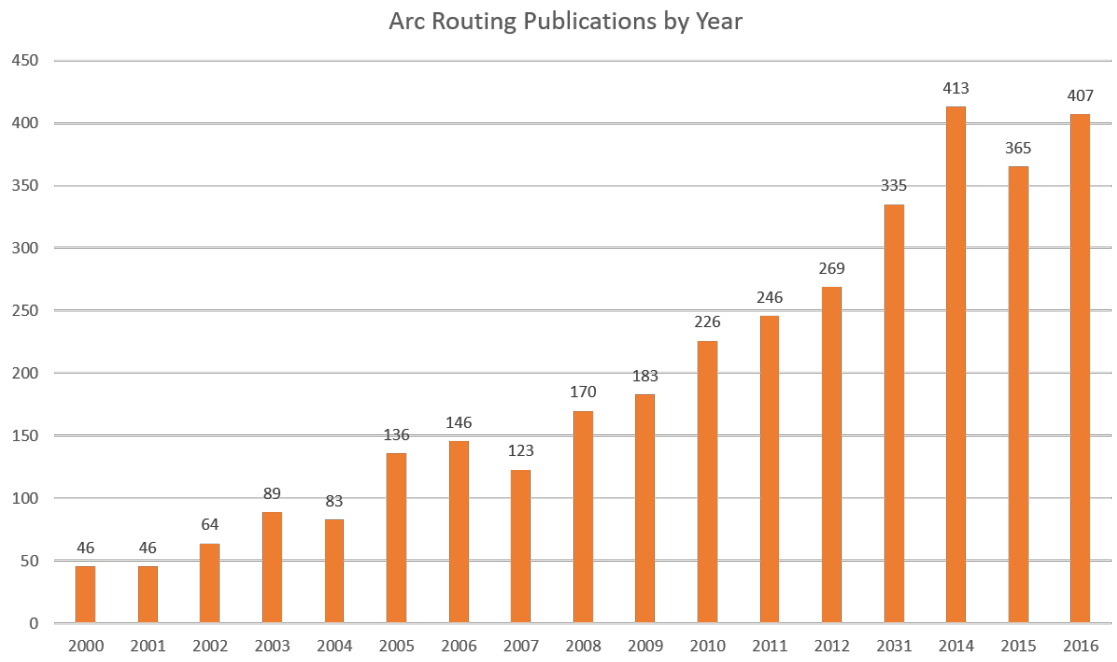
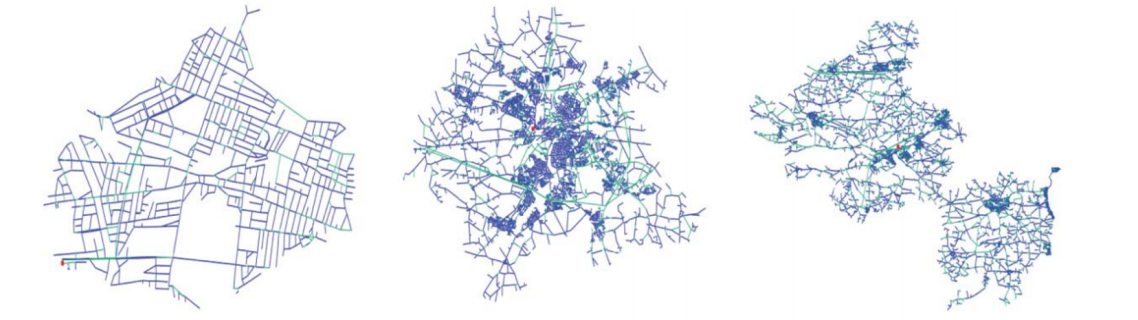


Figure 3.1: The number of articles on "arc routing" by year according to Google Scholar (May 20, 2017).

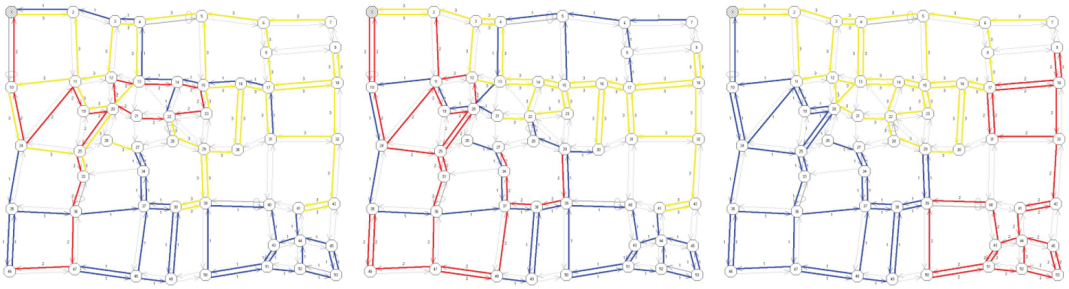
allows software developers to query it for map data [37]. A variety of geographic information systems (GISs) are built using OSM [38–41]. We develop a software tool called OAR Bench (Open-source Arc Routing Benchmark Instances Generator) that allows users to generate arc routing instances directly from map data taken from OSM. OAR Bench gives the user the ability to edit the generated instances by hand, or by using configurable parameters. The instances can be exported for use by researchers. In addition, OAR Bench has a visualization capability that can produce images of routes overlaid on the instance.

3.2 Literature Review

There are two primary approaches to generating benchmark instances in the routing literature: (1) GIS-based network generation, and (2) random network gen-

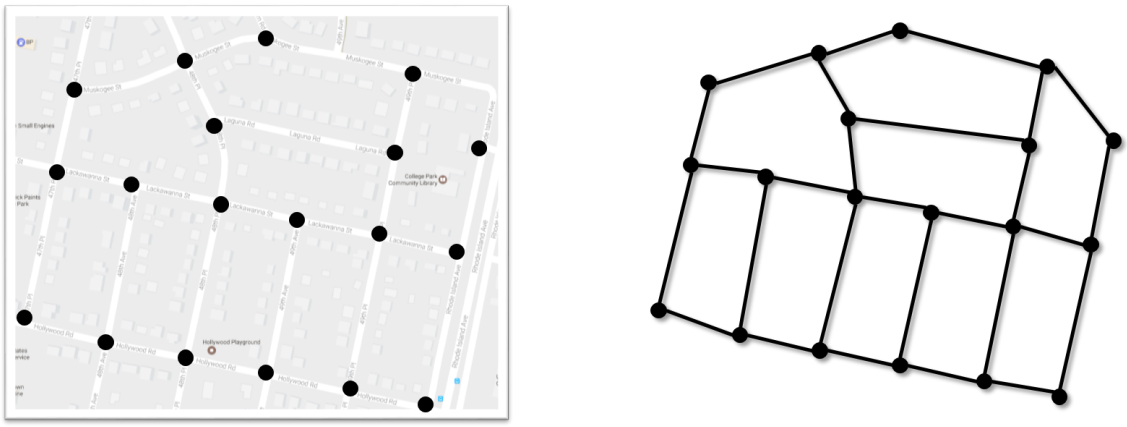


(a)



(b)

Figure 3.2: Three GIS-based benchmark instances developed by Kiilerich et al. [42] are shown in (a). The instances adhere strictly to a grid structure. Three partitions of a random network from Constantino et al. [43] are shown in (b).



(a) (b)

Figure 3.3: Generating a GIS-based instance. In (a), nodes are overlaid on a map of a neighborhood. In (b), nodes are connected according to the topology of the streets in the underlying neighborhood.

eration. In GIS-based network generation, an actual street network is used for the underlying graph. Additional problem features (e.g., which edges require service, service times, vehicle capacities, fleet characteristics, depot locations, etc.) are then overlaid on the network to create an instance. To generate multiple instances of different sizes from the same network, subsections of the network can be used. Examples of GIS-based network generation are given in [36, 44, 45]. In Figure 3.3, we show the process for generating a GIS-based instance. The structure of the GIS-based instance is very similar to the actual network encountered in practice. Good performance on the GIS-based instances should provide compelling evidence that a solution procedure might be useful in practice. However, it may be time consuming to get the required data to generate many instances, and to make sure that congruence with the actual street network is not lost when editing an instance or including additional problem features.

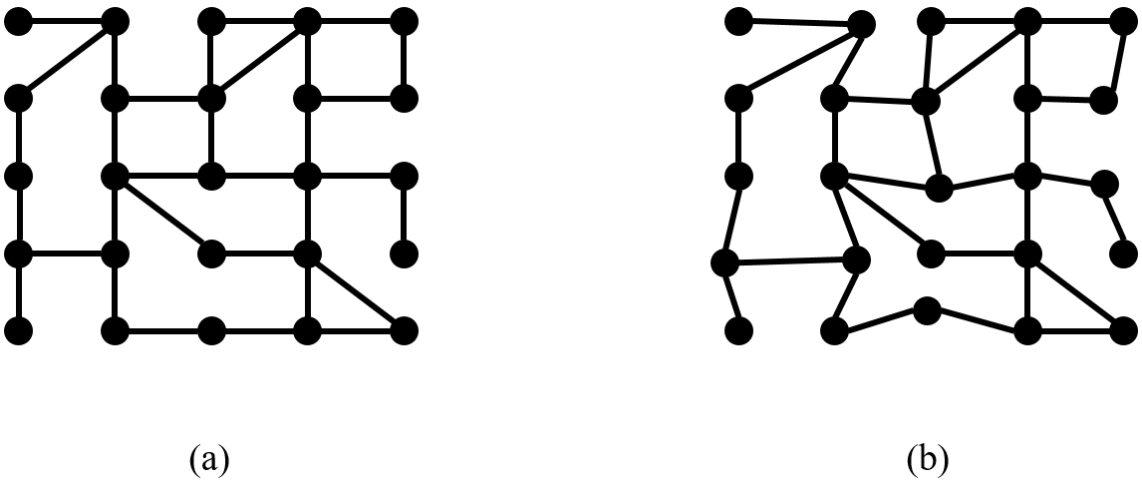


Figure 3.4: Generating a random instance. In (a), a 5x5 grid of nodes is randomly connected to adjacent nodes. In (b), the network from (a) is perturbed by randomly displacing some of the nodes.

In random network generation, the underlying graph is generated according to a common network model (e.g., Manhattan [46], Erdos-Renyi [47], Barabasi-Albert [48]). Examples of random network instances are given in [49, 50]. In Figure 3.4, we show the process for generating a random instance. With random instance generation, a large number of instances can be generated very quickly and easily in many sizes with different problem features. However, performance on these instances may not provide meaningful insight into real-world performance if there are significant differences between a random instance and the street network that practitioners encounter.

We point out that there are three well-known sets of randomly generated benchmark instances in the arc-routing literature: (1) *kshs* set in Kiuchi et al. [51], (2) *lpr* set in [52], and (3) *gdb* set in Golden et al. [53]. These three sets of instances are randomly generated. The six instances in *kshs* are based on complete networks

and have 15 edges and six to 10 nodes. The 15 instances (five small, medium, and large instances) in *lpr* are based on sparse networks and have 28 to 401 nodes and 52 to 1056 edges. The 23 instances in *gdb* are based on sparse networks and have seven to 27 nodes and 11 to 55 edges.

There are four well-known sets of instances which are based on real networks: (1) *egl* set in Eglese et al. [54], (2) *egl-large* set in Brandão and Eglese [55], (3) *bmcv* set in [56], and (4) *bccm* set in Benavent et al. [57]. The 24 instances in *egl* and 10 instances in *egl-large* are based on the streets of Lancashire, England. The *egl* instances have 77 to 140 nodes and 98 to 190 edges. The *egl-large* instances have 255 nodes and 375 edges. The 100 instances in *bmcv* are based on the streets of Belgium and have 26 to 97 nodes and 35 to 140 edges. The 34 instances in *bccm* are based on the streets of Valencia, Spain and have 24 to 50 nodes and 34 to 97 edges. For all of these instances, edge costs are based on the length of the streets.

The project by Zeni et al. [36], called VRP Bench, is relevant to our work. The authors use the Brazilian city of Artur Nogueira to produce benchmark instances for a multiobjective problem involving contributions from average route length and route imbalance. The underlying street network is extracted manually to mitigate issues with missing data. The density of required edges is a function of several variables including distance to the city center, type of road, and zoning (i.e., whether a road contains commercial, industrial, or residential buildings). One hundred benchmark instances with 1,000 to 10,000 delivery points were generated. The authors note that, in many cases where manual extraction of the network is not tractable due to the large amount of manual effort required, open-source map systems like

OSM can help automate the process of generating instances.

3.3 The OAR Bench Tool

OAR Bench is a desktop application that we developed using a combination of Javascript (JS), Hypertext Markup Language (HTML), and Cascading Stylesheets (CSS). Our code is combined and packaged using NodeWebkit Javascript [58] and has a portable browser that uses the same layout engine that powers Google Chrome and Apple Safari. This browser allows our application to be easily compatible with all major operating systems. Software dependencies are minimized. Our code is simplified by avoiding browser-specific tuning. The JS, HTML, and CSS software stack are standard technologies used in web development. They offer many visualization libraries that are well-suited for GIS. The visualization libraries that we use are listed in Table 3.1.

Leaflet is a Javascript library that enables interactive, tiled maps to be embedded within Javascript applications [60]. Users can navigate and zoom with a simple click-and-drag interface. Imagery is downloaded according to an on-demand streaming model where only tiles within, or near the edges of the frame, are requested to reduce the amount of data that has to be transmitted. An example of the Leaflet interface is shown in Figure 3.5. Leaflet allows a user to specify and preview a geographic bounding box that contains the street network used to create the instances.

Data-Driven Documents (d3) [61] is a widely used data visualization library for Javascript. d3 supports a variety of animations and presentation formats in-

Integrated Open Source Libraries		
Library	Description	Reference
Node-Webkit Javascript (NWjs)	The framework that enables the use of web development technologies to be used to create a local desktop application. This grants Javascript the ability to run additional modules, and packages a lightweight web browser with the code that makes the tool portable across the major operating systems.	[59]
Leaflet	Provides the map interface for selecting the geographic region containing the street network to be used in generating an instance.	[60]
Data-Driven Documents (d3)	Works with Leaflet to provide an overlay on the map of the queried region. This overlay appears when the user queries the map database for a size estimate.	[61]
Cytoscape	Provides the graph visualization interface in the refinment phase of the tool, where streets are selected to require or not require service.	[62]
jQuery	Provides document object model (DOM) parsing and more compact object referencing.	[63]
(Twitter) Bootstrap	Provides the user interface (UI) components for the navigation menus in the tool.	[64]
qTip	Provides the popup functionality used to display street properties when the user clicks on an edge on the Cytoscape canvas.	[65]
Vex	Provides the UI for inputting percentages used when randomizing the required streets in the instance.	[66]
Alertify	Provides notification functionality to convey error messages and status updates.	[67]

Table 3.1: Libraries used in OAR Bench.

cluding charts, tables, and graphs by providing a unified framework to bind data to established object models in a way that simplifies data reuse. For example, data can be stored in Javascript data structures, and then visualized as an HTML table or a scalable vector graphic (SVG) histogram without having to write custom HTML and SVG that would otherwise be required. A d3 powered graphic is shown in Figure 3.6. In the context of OAR Bench, we use d3 to render an SVG overlay of the streets that will be exported on top of the Leaflet map.

Cytoscape is an interactive graph visualization capability that contains native support for click-and-drag navigation, selection of nodes and edges, some basic graph

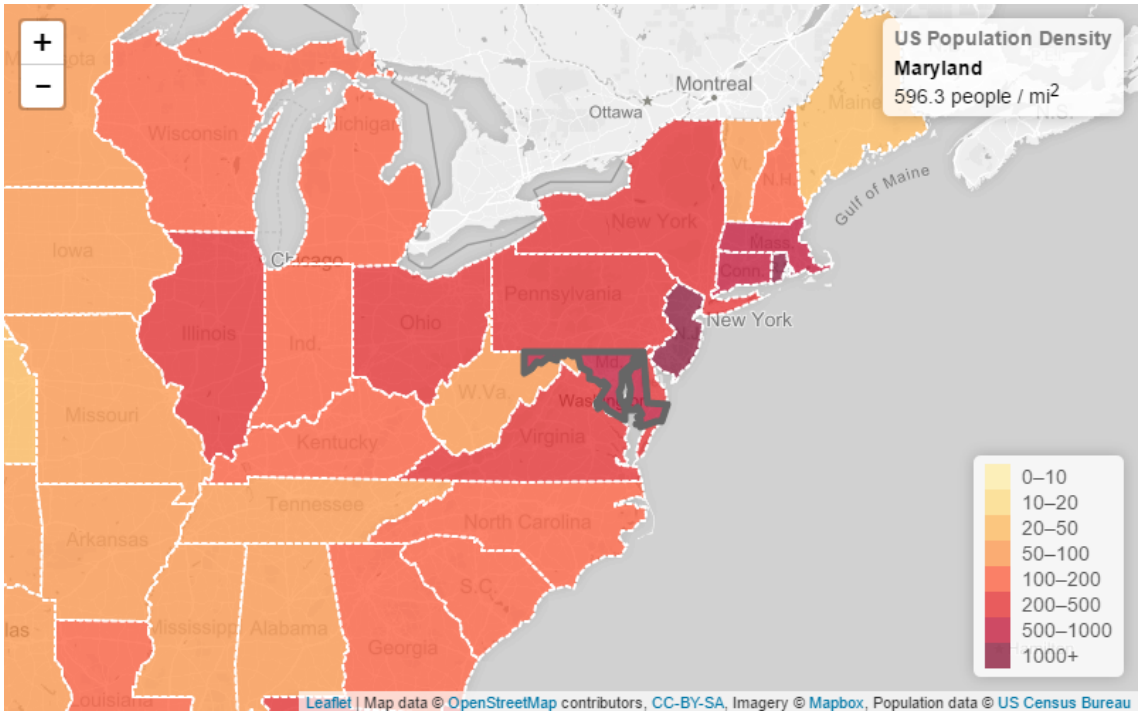


Figure 3.5: A map application that uses Leaflet to visualize the population densities of each state in the United States [68].

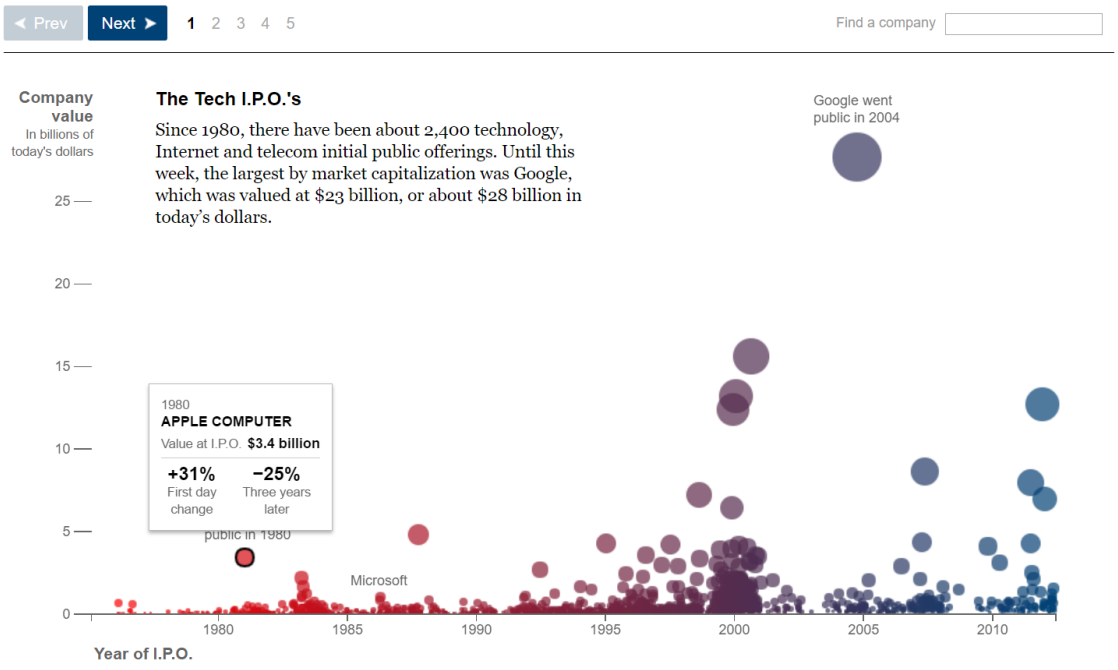


Figure 3.6: An interactive graphic powered by d3 from The New York Times website on May 17, 2012 [69]. The graphic depicts the value of a tech company prior to its IPO.

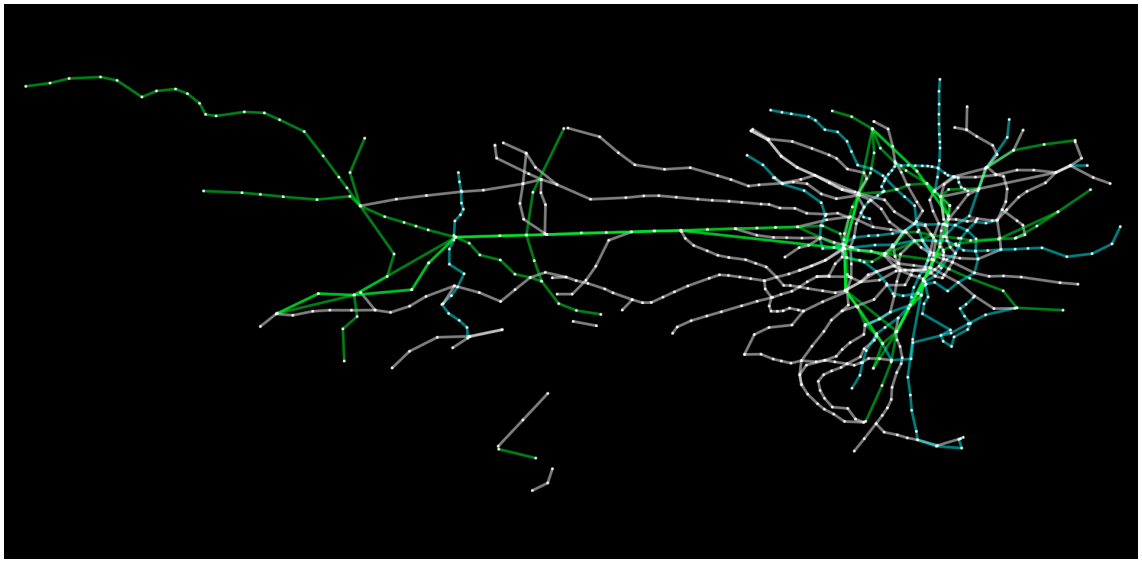


Figure 3.7: A map of the Tokyo railway system recreated using Cytoscape [76].

algorithms (e.g., shortest path, connected components), and a variety of styling options [62]. Cytoscape is available as a standalone application and a Javascript library. Cytoscape is primarily used in biological work [70–73] and in other applications such as the study of authorship networks and networks with semantic data [74, 75]. A visualization of the Tokyo metro map produced by Cytoscape is shown in Figure 3.7. In OAR Bench, we use Cytoscape as the engine that powers the interface used to modify and refine the raw street network.

We use several common Javascript utility libraries to ease coding and improve usability including jQuery [63], Bootstrap [64], qTip [65], vex [66], and alertify [67]. In our work, jQuery enables less verbose object referencing, while Bootstrap, qTip, and vex provide various elements of the UI. Alertify provides functionality to enable the pop-up notifications and system alerts.

We use two phases to produce an arc routing instance in OAR Bench: (1) generate phase, (2) refine phase. In the generate phase, the user specifies a geographic

bounding box using the Leaflet map interface. The user has the option to consider the entire region contained within the frame or to specify a subset of the region. After this area is selected, the user can query the Open Street Maps database and then have the selected street network overlaid on the map. This allows the user to preview the selection before confirming it. After the user is satisfied with the selection, street data may be exported to a Javascript Object Notation (JSON) file that will be used in the second phase.

In the refine phase, the user can load the street network into a Cytoscape canvas and make manual edits to the instance. The user may remove nodes and edges in the graph, mark certain nodes as depots, and select edges to be marked as requiring service. In order to expedite and simplify the process of marking required edges, we allow the user to specify what percentage of the streets are required based on the metadata contained in the JSON file exported from the generate phase. This metadata includes the priority of the street (e.g., primary for highways, secondary for major roads and thoroughfares, tertiary for local streets) for use in problem variants where critical infrastructure must be serviced before local roads ([77–79]), the speed limit for determining traversal times, whether or not the street is one way, and what types of buildings are located on the street (which we call the zone of the street). For example, the user could generate instances that have 30% of residential streets, 10% of tertiary streets, and 50% of streets that aren't labeled with priority data. In addition, the user could specify that 20% of commercial streets and 30% of streets on university campuses be required.

3.4 Example Instances and Usage

In this section, we describe several ways to use OAR Bench to generate test instances. In the first example, we consider a package delivery scenario. We use this example to follow the OAR Bench workflow. In the second example, we generate two related instances, where one instance has a subset of the streets removed to model an event such as a natural disaster that has edges in a portion of the network unavailable for use. In addition, the user wishes to customize the original OSM database request made in the first phase and not include streets with a priority less than tertiary. In the third example, there is a complex demand profile that combines all of the metadata options. For a detailed example of generating an instance based on a portion of the streets of Amsterdam, see Appendix C.

In the first example, we describe the process of generating a street network by manually specifying the streets that are required and exporting an instance. The process begins with the user in the Generate tab of OAR Bench with the embedded Leaflet map. The UI is shown in Figure 3.8. At the top right of the map, the user can toggle between satellite imagery and street map tiles. We show satellite imagery in Figure 3.8.

In order to generate the street network file, the user must specify the geographic region of interest. By default, the entire area on the screen is designated as the region of interest. The user may zoom and pan to bring the desired area onto the screen. If the user wants to specify a different bounding box (e.g., with a different aspect ratio), the button on the left of the map allows the user to select a subset

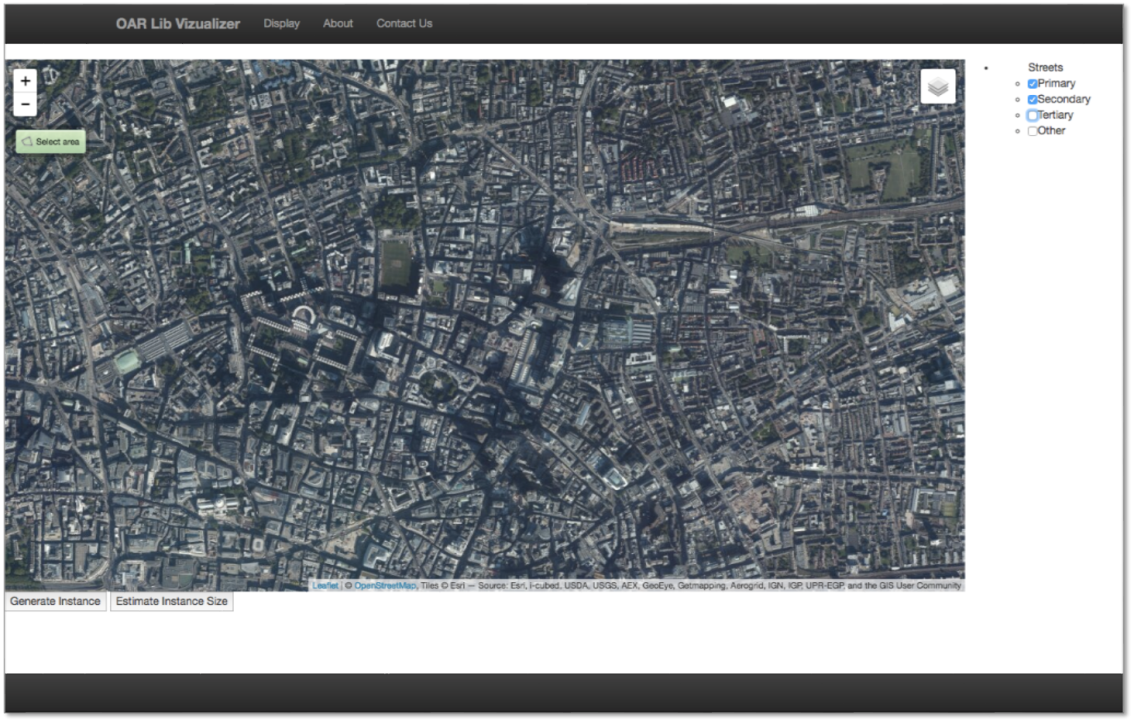


Figure 3.8: The OAR Bench UI for the network generation phase. The Leaflet map pane is used to select the geographic region that provides the underlying street network. Satellite imagery and a cleaner street map layer are available.

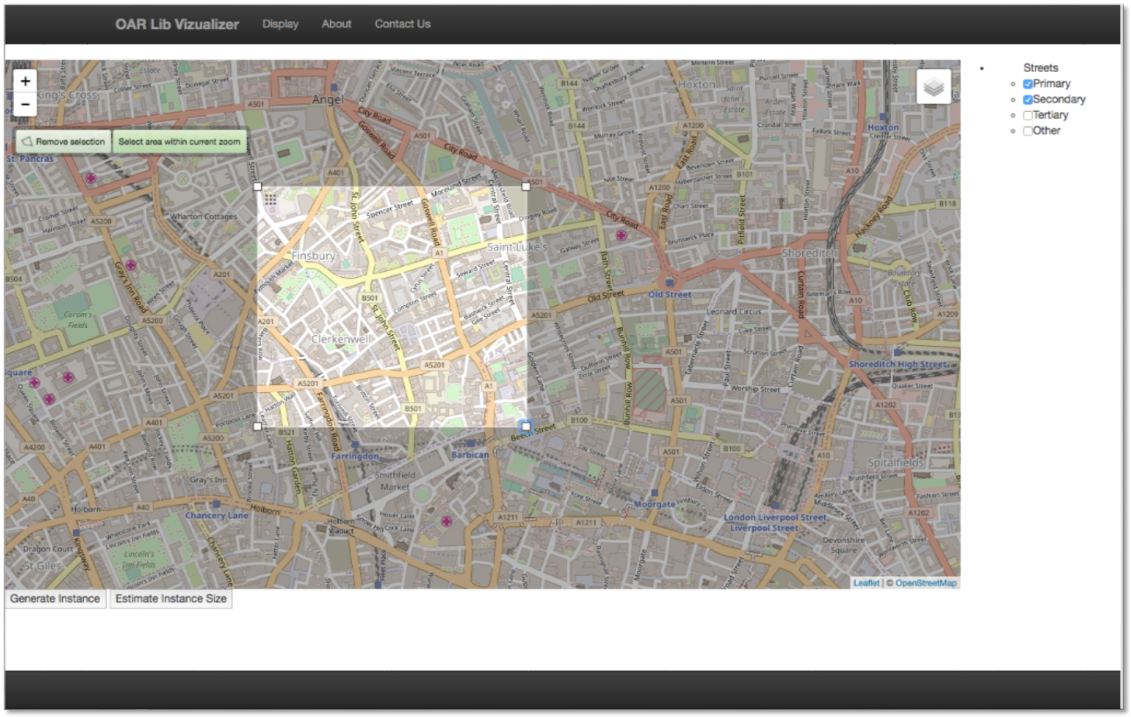


Figure 3.9: The user can query OSM for a subset of the visible area by specifying a bounding box.

of the region displayed on screen. This is shown in Figure 3.9. At any time in the Generate page, the user can estimate the size of the instance that would result from exporting the instance with the current settings. Estimating the instance size (i.e., number of edges) overlays the streets that would be exported on top of the Leaflet map, as shown in Figure 3.10. In the context of OAR Bench, each node represents an intersection and edges are the street segments that join them (i.e., a single street is usually represented by many edges).

After the network has been exported, the user has the option of using the file and specifying instance details (e.g., which streets are required, where the depot is located). The file can also be imported into the Display tab of the application. Like the Leaflet map, the Cytoscape canvas allows zooming and panning. The UI

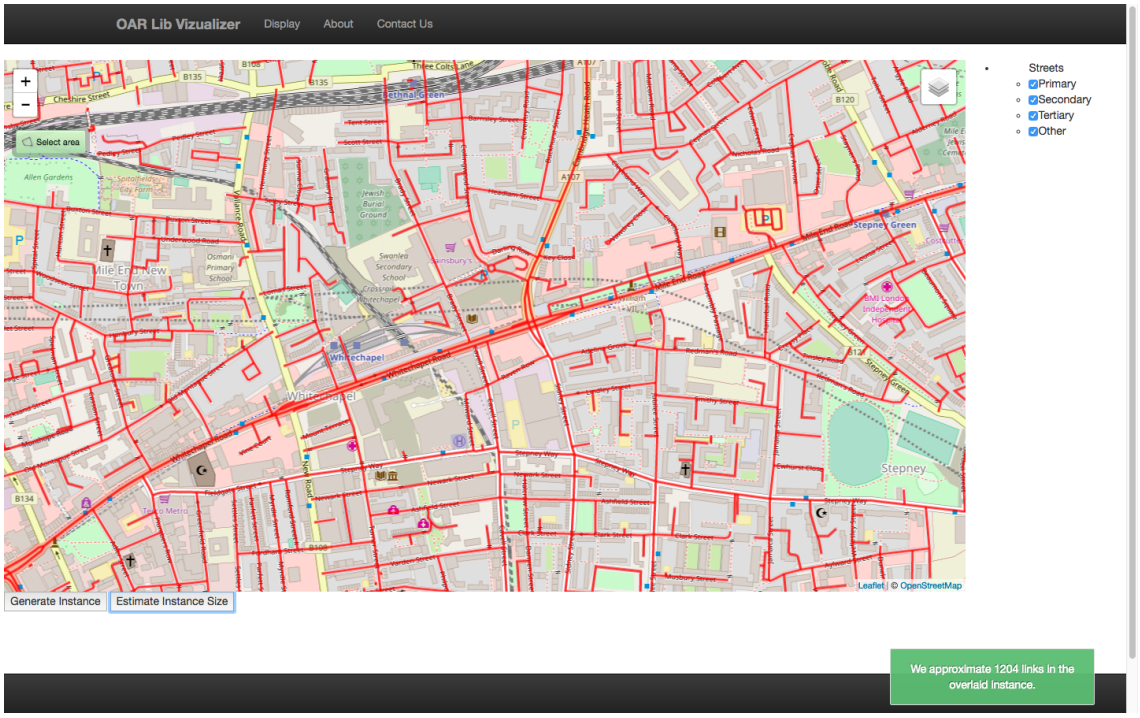


Figure 3.10: If the user wants to see which streets are included in the export, and get a rough estimate of the size of the instance, the database can be queried with the Estimate Size button. The streets will be overlaid in red on top of the map, with an estimate of the number of edges.

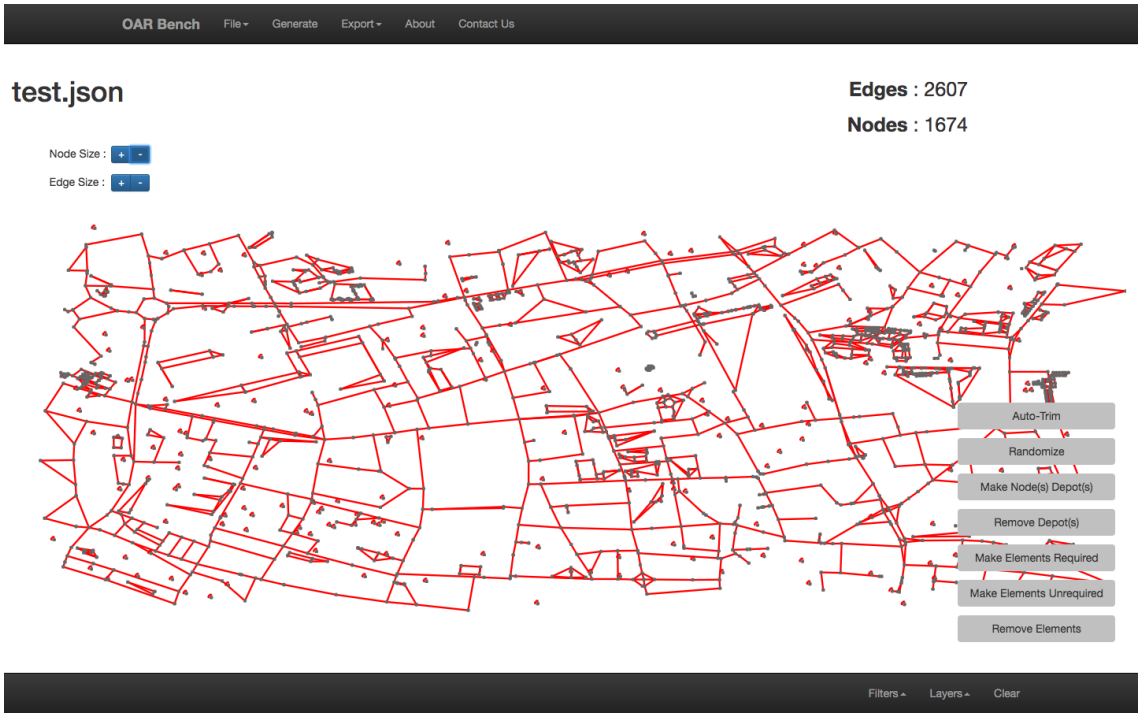


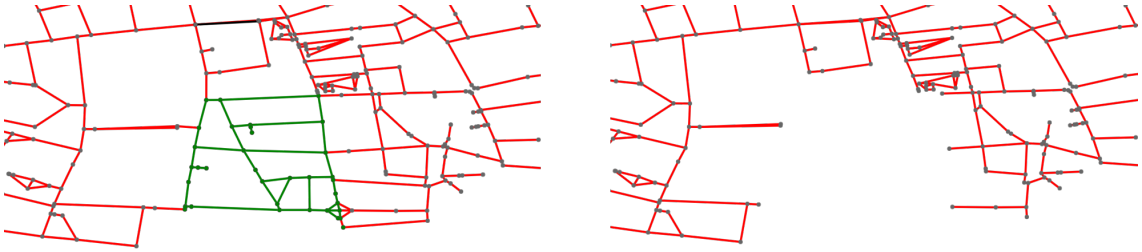
Figure 3.11: The OAR Bench UI for the display and refine phase for generating an instance. A network has been imported from the generation phase. Initially, edges are colored red to indicate they do not require service.

is shown in Figure 3.11. On this page of the application, the user can select streets and mark them as required and can select some nodes and mark them as depots. In addition, there is an Auto-trim button that will remove streets that are not part of the largest connected component of the graph. Since the network produced in the Generate tab includes all streets that are contained in the region of interest, there will be some streets that may not be connected to the rest of the network, and therefore are unreachable in a routing context. A before trimming view and after trimming view are shown in Figure 3.12 for a section of a street network. The final instance can be exported to a text file, where the distance of an edge is proportional to the length of the street.

In the second example, the user generates several variants of the same instance

(a) (b)

Figure 3.12: The effect of trimming an instance immediately after importing it from the generate phase is shown. In (a), we see the network from OSM and a zoomed-in section of the map. In (b), after trimming, all unconnected edges and nodes have been removed, leaving only a single connected component.



(a) (b)

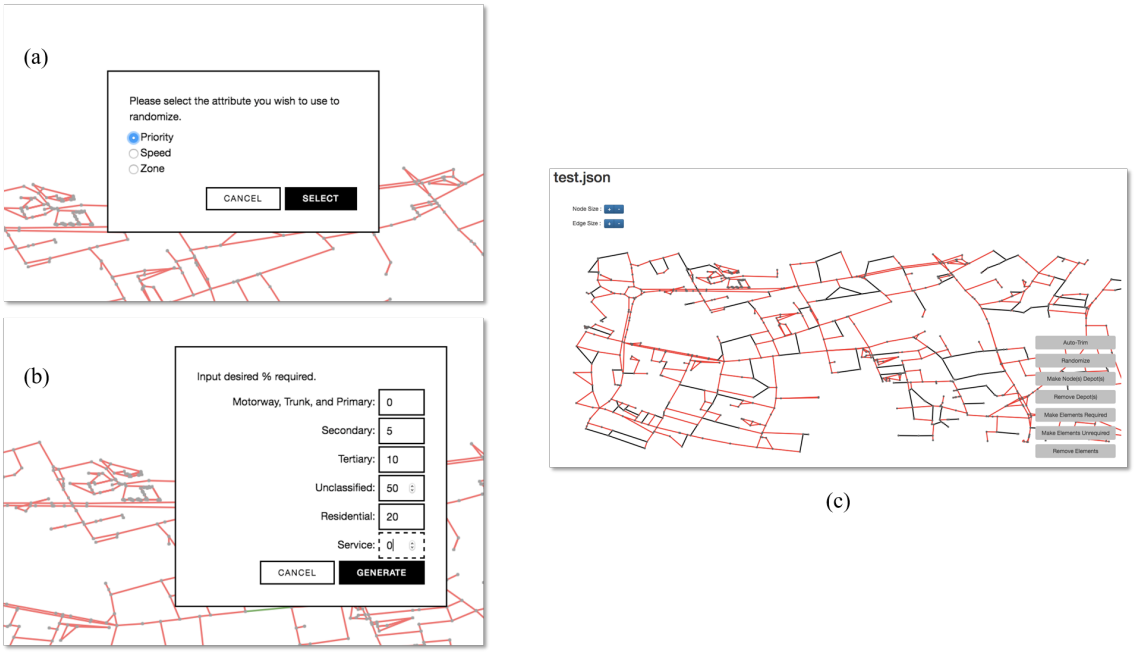
Figure 3.13: Manually selecting and removing a subset of streets. The streets and nodes in green on the left are removed, resulting in the network on the right.

where sections of the network have been removed. The removed sections may be road closures on routing operations or damaged infrastructure following a natural disaster [80–83]. After the street network is imported into the Display page, the user can select street segments and nodes and use the Remove Selected Elements button to delete them. The Auto-trim button can then be used to clean up parts of the graph that may have become disconnected as a result of removing selected elements. An example of the removal procedure is shown in Figure 3.13.

In the third example, the user wants the street network to have streets with only tertiary priority or above. The user would like to specify the percentage of required streets according to priority, maximum speed, and zone. This example is very similar to the problem considered in Zeni et al. [36]. To accomplish this, in the Generate page, the types of streets to include in the exported network can be specified. By default, all streets are included in the export. However, depending on the region, the network can include many side streets that are not of interest. Therefore, OAR Bench allows the user to select streets with particular priorities to include in the export. An example is shown in Figure 3.14. In the Display page of OAR Bench, there is an option to randomize the required streets in the network. Percentages can be specified according to each of the metadata tags (priority, speed limit, zone) that are embedded in the street network file. For larger instances that have hundreds or thousands of edges, specifying required edges based on these tags can be a much faster way of marking streets as required. This process is shown in Figure 3.15.

(a) (b)

Figure 3.14: During the generate phase, there are filters (highlighted in the red box) that can be set to include or exclude certain types of data from the OSM query. In (a), all street priorities are checked. Exported streets appear in red. In (b), unchecking all priorities below secondary excludes those streets from the generated network. The result of using filters is reflected in the red overlay from estimating the instance.



(c)

Figure 3.15: For medium- and large-size instances, manually specifying streets that require service can be a tedious process. To mitigate this, OAR Bench has the capability to randomly assign streets as requiring service according to their properties. In (a), street priority is used. In (b), the percentage of streets that is required for each street priority is specified. For example, in the box marked Secondary, we are requesting that 5% of streets with a priority of secondary be required. It is worth noting that the percentages do not need to sum to one since they are independent. In (c), the resulting network with required streets colored black and the rest of the streets colored red is displayed.

3.5 Limitations of OAR Bench

OAR Bench has several inherent limitations. First, there can be a large amount of missing data. All information in the OSM database is contributed by users, so the database relies on motivated members of the GIS community to populate it. While most streets and street names are included in the database, especially in well-populated areas, information about priorities, speed limits, and zoning can be limited. This means that a large number of streets may have undefined or unknown fields in the metadata. This limitation is common to most GIS systems that use free public maps [36].

A second limitation is the scalability of the street network visualization. While OAR Bench's display capability has been tuned and is able to handle thousands of graph elements while still maintaining reasonable performance (i.e., graphical rendering is fast enough to keep up with panning, zooming, and other interactivity options), the utility of the visualization can decrease when the network size is large (approximately greater than 2000 graph elements, where a graph element is a node or an edge). While the user can zoom in and out, details of the network can become difficult to see on screen when zoomed out. Filtering mechanisms that allow the user to toggle the visibility of edges according to their properties, mitigate the second limitation by decluttering the screen and allowing the user to see the structure of the network (e.g., where arterial streets are, where different zones are). Network visualization remains an area of open research, even when graph elements can be repositioned freely [84–86].

A third limitation involves finding street networks with specific properties (other than network size). For example, if the user wants to create instances with a particular network density, or a particular average degree, the only way to accomplish this currently would be to estimate these properties from the map imagery. The user might want to do this in order to analyze the effect of these properties on the performance of a solution procedure. However, if the user wants to conduct this type of analysis, random networks can be generated with the desired characteristics and, therefore, we feel this is not a significant shortcoming of OAR Bench.

3.6 Conclusions

In this paper, we presented a new software tool (OAR Bench) that generates arc routing instances from an open-source map database. Our software tool uses a variety of information about the street including street priority, speed limit, and street type that allows researchers to tailor the instances to their particular problem. Our software tool also allows users to manually edit instances to remove sections of the network, designate nodes as depots, and designate edges as required. With the capabilities and flexibility of OAR Bench, users can create a wide variety of new instances that capture features found in real-world street networks. In addition, users can overlay images of vehicle routes on an instance. This allows users to quickly diagnose the compactness, contiguity, and separation of routes.

Although OAR Bench was designed specifically for arc routing problems, other network applications may be able to benefit from using it. For example, the same output could be used to construct realistic instances for time-dynamic shortest paths

or network flow problems. Streets with high priority may have more variable traversal times relative to residential or service streets which will never be as well-traveled. Speed limits can be used to infer more accurate travel times. The visualization and network editing capabilities can be used to understand the effect of road closures.

In future work, we hope to extend OAR Bench to node routing problems by including data for building locations found in open-source map databases.

Chapter 4: A Hybrid Heuristic Procedure for the Windy Rural Postman Problem with Time-Dependent Zigzag Service

4.1 Introduction

In arc routing applications, some streets may require service along both sides of the street. We refer to this as zigzag service and illustrate it in Figure 4.1. In contrast to servicing each side separately, the vehicle stops more frequently and incurs a traversal cost, two service costs, and a penalty cost associated with the slowed travel time required to perform the zigzag service. The tradeoff is that the vehicle only needs to service the street once on its route. For some streets, the zigzag service is only possible during the early morning hours when there is very little traffic. This scenario is modeled by the Windy Rural Postman Problem with Zigzag Time Windows (WRPPZTW). Time window constraints are present in many routing applications [87–91]. Typically, these constraints require certain customers to receive service during a specified time of the day. In this problem, the time of day restricts the service options that are available to a vehicle. By considering these additional service options, it is possible to reduce the cost of a route, as shown in Figure 4.2. In Figure 4.2(a) a dashed arc does not require service and the traversal cost is shown next to the arc. The solid arcs all have a traversal cost of $D = 10$, a service cost of $S = 5$, and a zigzag cost of $Z = 12$ that includes the service cost and

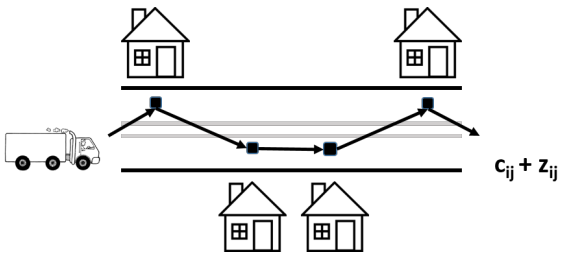


Figure 4.1: The zigzag service. The truck is traversing the street from the left to the right. Typically, the truck would drive on the bottom lane and service the two houses. However, it now has the option to zigzag and service all four houses during this traversal. The traversal cost is the sum of the usual traversal cost (c_{ij}) and a specified zigzag cost (z_{ij}).

the penalty cost. In Figure 4.2(b), the optimal solution is shown with no zigzagging. Each of the three streets that require service on both sides incur a cost of 30 (15 per side). Street $(1, 2), (2, 1)$ has a total cost of $15+5 = 20$. In Figure 4.2(c), the optimal solution is shown when zigzagging is allowed on streets $(2, 3)$ and $(3, 4)$. The cost of the optimal solution drops from 110 to 97. In Figure 4.2(d), the optimal solution is shown when the street $(0, 1)$ may be zigzagged only during the first 25 time units of the planning period. The objective value further decreases to 93. In Figure 4.2(e), we show that zigzags with time windows can change a route dramatically. When street $(3, 4)$ has a zigzag option during the first 35 time units of the planning period, we have a very different route compared to Figure 4.2(d).

Two factors motivate the development of a heuristic. First, we know that the integer program for the WRPPZTW given in [92] struggles to solve problem instances with tight time windows. However, tight time windows limit the number of possible solutions that could be considered by a heuristic method. Furthermore, after the time windows, the remaining part of the problem is a Windy Rural Postman Problem with Zigzags (WRPPZ) and it becomes much more computationally

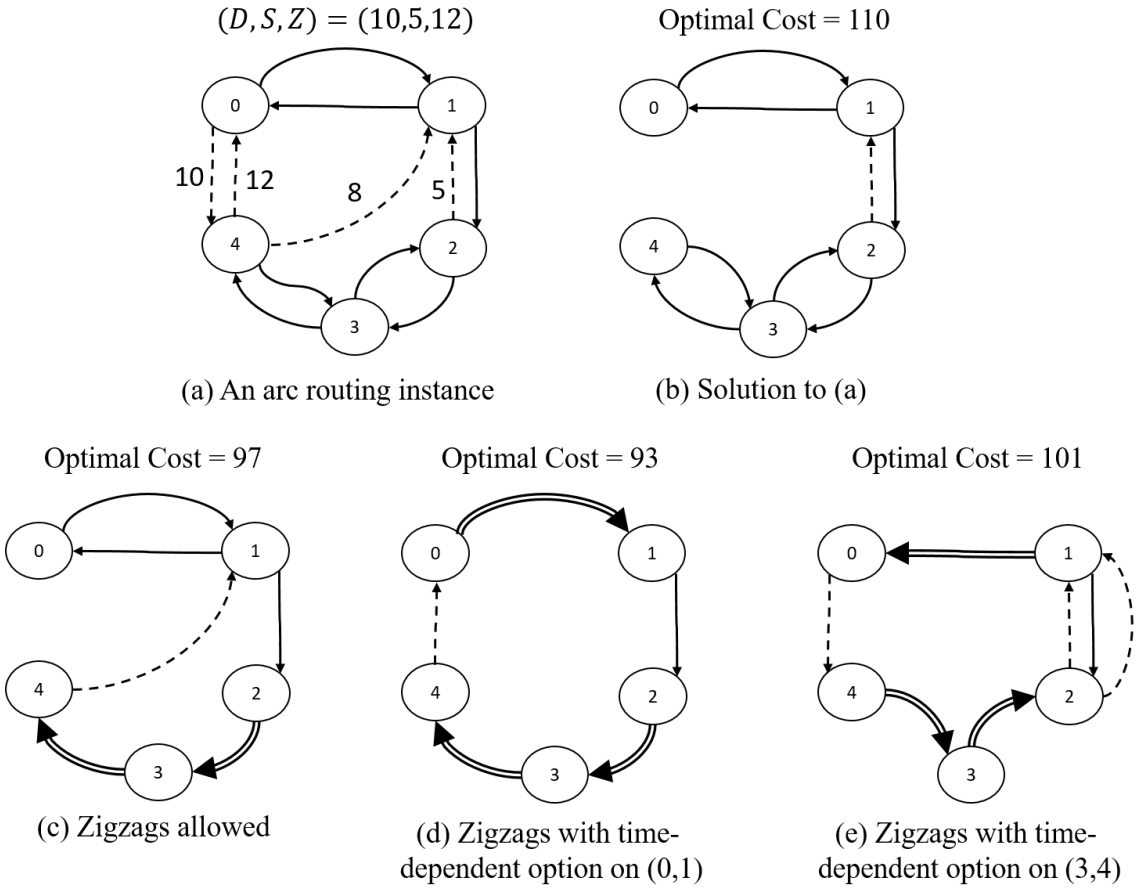


Figure 4.2: An example that shows how zigzagging can impact solution quality.

tractable. In this chpater, we formulate an integer program (IP) that is capable of producing an exact solution to the WRPPZ quickly. Therefore, we focus on problem instances where time windows are tight and imposed only during the early portions of the planning horizon and decompose the problem into two phases. In the first phase, we use a heuristic method to find a partial route that satisfies the time windows. Then, in the second phase, the partial route from phase one is fixed and used as input for the IP model. Solving the IP model for the WRPPZ gives a complete solution. We expect that a heuristic in which the portion of the route occurring after the zigzag deadline is determined optimally should be effective.

In Section 2, we formally define the WRPPZTW. In Section 3, we provide a brief review of the relevant literature. In Section 4, we present our heuristic, including a new scalable formulation for the WRPPZ that is used in the second phase of our heuristic. In Section 5, we present computational results comparing the solution quality and computing time of our heuristic to the results from the branch-and-cut procedure described in [92]. In Section 6, we summarize our work, provide concluding remarks, and give directions for future work.

4.2 Problem Definition

The WRPPZTW is defined on a street network $G = (V, E)$, where V is the set of vertices v_i and E is the set of edges e_{ij}. The cost of traversing e_{ij} is given by c_{ij}. The graph is windy, so $c_{ij} \neq c_{ji}$, in general. The subset $E_R \subseteq E$ contains edges that require service. A required edge requires service in at least one direction.

Therefore, it may be the case that $e_{ij} \in E_R$ and $e_{ji} \notin E_R$. Each required edge has one of three possible zigzag states: (1) zigzag required, (2) zigzag optional, or (3) zigzag prohibited. These three states are shown in Figure 4.3. In Figure 4.3(a), if a vehicle traverses the street in both directions on its route, it must service all customers along both sides of the street in one traversal. In Figure 4.3(b), we need to determine whether the street receives zigzag service or two-pass service. In Figure 4.3(c), zigzag service is not possible. A vehicle must traverse the street once in each direction (not necessarily in immediate succession) in order to service customers along the route. In Figure 4.3(d), each street that allows zigzag service (either zigzag required or zigzag optional) may have a time window. Outside of a time window, a vehicle must service customers on each side of the street separately. Every required edge (regardless of zigzag state) has a service time s_{ij} that is incurred only when a vehicle services a street. A required edge may have different service times s_{ij} and s_{ji} for each direction that requires service.

If zigzagging is not possible for a given street, then each required direction must be serviced separately. That is, if there are customers on both sides of the street, the service vehicle must traverse the street at least once in each direction. If zigzagging is required, then, even if a route has a vehicle that traverses the street in both directions, it must incur the zigzag cost when servicing the street. If the street is zigzag optional, then we must decide whether a vehicle services customers on the street with two-pass service or with zigzag service.

If it is possible to zigzag an edge, there is a time window $tw_{ij} = tw_{ji} = [a_{ij}, b_{ij}]$ associated with the edge, where a_{ij} is the earliest time and b_{ij} is the latest time

that a vehicle may zigzag this edge. Setting $a_{ij} = 0$ and $b_{ij} = M$, where M is arbitrarily large, indicates that there is no time window on an edge. In this chapter, we consider time windows with $a_{ij} = 0$. This reflects the real-world environment in which zigzagging could only happen in the early morning. A zigzag optional edge has a zigzag cost z_{ij} that is incurred if the edge is zigzagged. We include service costs in the zigzag costs so that $z_{ij} = s_{ij} + s_{ji} + penalty_{ij}$, where $penalty_{ij} > 0$ represents any additional time required for the vehicle to perform the zigzag service. The cost of servicing both sides of a street via zigzagging in the direction from i to j is then $c_{ij} + z_{ij}$. A feasible solution to the WRPPZTW is a route r through G that services all required edges and satisfies the zigzag time windows. The optimal solution minimizes the sum of the traversal costs, service costs, and zigzag costs. A summary of our notation is given in Table 1.

4.3 Literature Review

The WRPPZTW combines several classes of network optimization problems: arc routing problems; problems with time windows, and problems with a zigzag service option.

There is an extensive literature on arc routing problems. Edmonds and Johnson [93] described one of the first, and most fundamental problems in arc routing, namely the Chinese postman problem (CPP). In the CPP, all streets in an undirected network require service. There are no further restrictions such as time windows, vehicle capacities, and service costs. The authors presented analytic results

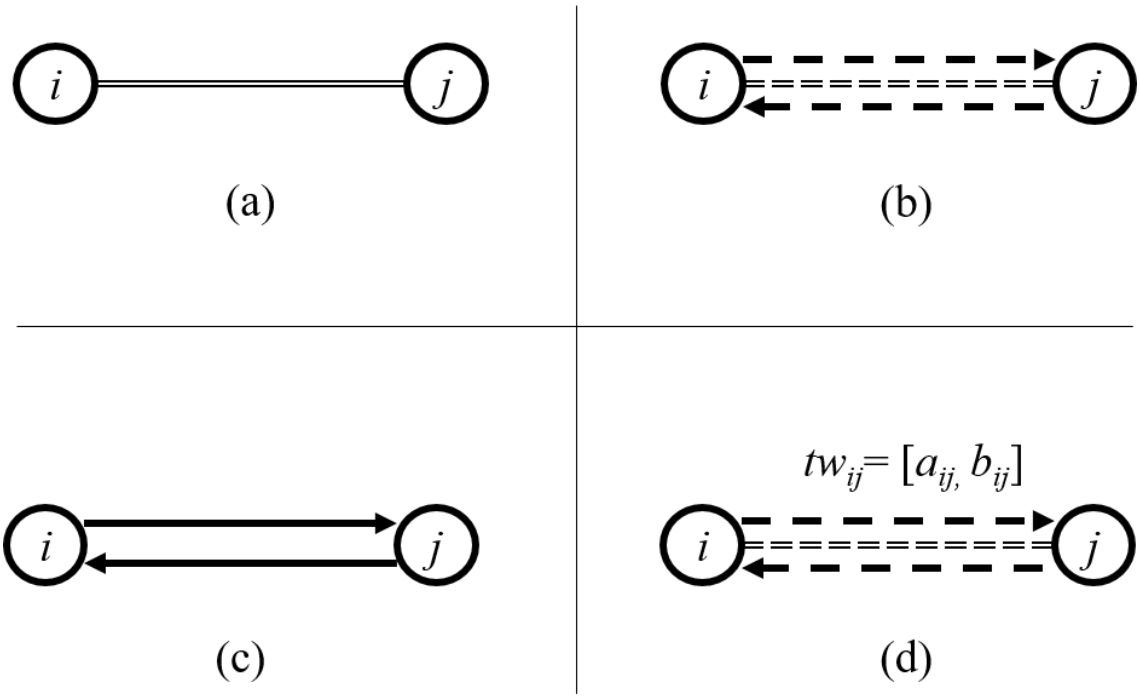


Figure 4.3: In the WRPPZTW, a street that requires service in both directions falls into one of three categories: (a) zigzag required, (b) zigzag optional, or (c) zigzag prohibited. In addition, zigzag optional streets may have a time window tw_{ij} outside of which the street may not be zigzagged, as shown in (d).

Notation	
Symbol	Description
G	Graph over which the problem is solved
V	Vertex set of G
E	Edge set of G
E_R	Subset of E containing only the streets that require service
v_0	Depot
v_i	Vertex i
e_{ij}	Edge from vertex i to vertex j
tw_{ij}	Time window for edge e_{ij}; only valid for edges that allow zigzag service
a_{ij}	Start time associated with tw_{ij}
b_{ij}	End time associated with tw_{ij}
c_{ij}	Traversal (deadhead) cost to traverse edge e_{ij}
z_{ij}	Zigzag cost to service edge e_{ij}; only valid for edges that allow zigzag service
s_{ij}	Cost for servicing customers without zigzagging while traversing the edge e_{ij} from i to j

Table 4.1: Notation for the WRPPZTW

and developed an exact, polynomial-time solution algorithm.

An NP-hard generalization of the CPP is the Windy Rural Postman Problem (WRPP). The WRPP was introduced by Benavent et al. [94] and does not include service time or the possibility of zigzag service. The authors formulated the problem as an integer program, characterized the polyhedron, and presented heuristics for

problem instances with thousands of nodes.

Christofides et al. [95] and Baker [96] are two of the first papers to describe time windows in network scheduling problems. The authors presented branch-and-bound procedures for the makespan problem with time windows. Savelsbergh et al. [97] showed that finding a feasible solution to the traveling salesman problem is NP-hard when time windows are considered. Golden and Assad [98], Solomon and Desrosiers [99], and Bräysy and Gendreau [100, 101] provided surveys of solution techniques for the vehicle routing problem with time windows (VRPTW).

Time windows were first applied to arc routing problems by Bodin and Kursh [88]. The time windows accounted for parking regulations in street sweeping. The authors developed a cluster-first, route-second heuristic, and a route-first, cluster-second heuristic that solved a subproblem for each set of parking constraints. Eglese [90] considered a capacitated vehicle routing problem with multiple depots, and several classes of streets with common time window constraints (e.g., requiring service within the first two hours of the planning horizon). He provided a cluster-first, route-second procedure within a simulated annealing framework. Aminu and Eglese [102] considered more general time constraints for the CPP in subsequent work. The authors developed a constraint programming approach based on a transformation into a node routing problem given by Pearn et. al. [103]. Reghioui et al. [104] studied the capacitated arc routing problem with time windows (CARPTW). They developed a greedy randomized adaptive search procedure (GRASP) heuristic for the CARPTW.

Arc routing problems with the zigzag service option are relatively new in the

literature. Irnich [105] considered postman problems where zigzag service is added to an undirected problem. A mixed integer program (MIP) modeled the problem. Irnich [106] presented a neighborhood search heuristic for a rural variant of a postman problem with zigzag service. We are aware of only one paper that considers the WRPPZTW. Nossack et al. [92] presented a MIP for the WRPPZTW by transforming the problem into a node routing problem. They used a branch-and-cut procedure to solve instances with 10 to 20 vertices.

Our method is based on a greedy push forward insertion heuristic (PFIH) introduced by Solomon [107] for VRPs with time windows. A PFIH procedure is similar to a savings-based initialization procedure. However, for each insertion at a position i, PFIH considers the effect on all customers at position $r > i$, checking for both feasibility and reductions in wait times. Cardoso et al. [108] presented an adapted PFIH that included an additional distance term and time term in the push first insertion cost.

4.4 Heuristic for the WRPPZTW

Our heuristic uses a push forward insertion method. In the first phase, we construct an ordering of the required edges in the graph. This order is assigned by giving a priority cost to each required edge based on its position relative to the depot. Next, a partial WRPP route is constructed by performing cheapest insertion, using the priority costs to determine the order of insertion. Finally, in the second phase, an integer program for the WRPPZ (without time windows) is used to complete

the route. In Figure 4.4, we outline the structure of our heuristic. The two solid arcs from the depot to the seed customer in the solution are determined by the initialization procedure. The three solid arcs form the initial partial route. This partial route is used to fix variables in the integer program for the WRPPZ that determines the remainder of the route (shown as dashed edges).

To begin the route, a seed street is chosen from the set of zigzaggable edges with time windows. The partial route $v_0 - e_{seed} - v_0$ is formed where e_{seed} is a randomly chosen seed customer, and v_0 is the depot. By varying the seed, we can generate many initial solutions. Then, each required edge that has a zigzag option and a time window is assigned a score according to the following function:

$$PFIHScore(e_{ij}) = -\alpha d(e_{ij}, v_0) + \gamma \Theta_{ij} \frac{d(e_{ij}, v_0)}{360}, (i,j) \in E_R \tag{4.1}$$

where d is the shortest path distance function, and Θ_{ij} is the angle in degrees between the depot and each customer on edge e_{ij}. Taking into account the angle and the shortest path distance between the depot and an edge e_{ij}, the endpoint that has a smaller shortest path distance is used. The parameters α and γ are tunable and, without loss of generality, sum to one. They assign relative importance to each cost component. A general version of the push first insertion method was introduced by Solomon [107]. The general procedure contains two terms that involve the beginning time and end time for each time window. In our problem, the beginning time and end time are the same for each time window, i.e., $tw_{ij} = [a, b]$, so we drop them. The α term gives higher priority to edges that are closer to the depot. The γ

term establishes a sweep ordering to the edges in order to take advantage of the approximately 2D geometry of a street network. For our heuristic, we use $\alpha = .7$ and $\gamma = .3$. These two values were empirically derived for a VRP variant with time windows [109].

Next, cheapest insertion is performed using this ordering. At each iteration of the insertion, the unrouted edge with the smallest score is selected for insertion into the route. During an insertion, it is possible that the inserted edge's time window is obeyed, but the insertion displaces another edge later in the route and may cause an infeasibility with respect to a time window. These moves are removed from consideration. If no feasible moves are available, the edge is removed and we consider the next edge in the ordering. When the insertion procedure is completed, we have a partial feasible solution. In Appendix A, we show how the partial route initialization procedure works.

Finally, we formulate a new integer program (IP) for the WRPPZ. This IP solves for the portion of the route after all edges with time windows are serviced (see Figure 4.4). Our new IP, which is different from a version of the IP in [92], is needed to scale to problem instances with hundreds of nodes. Our IP is given in Section 4.1.

Our heuristic generates many initial partial solutions by varying the seed street used during the insertion phase of initialization, and runs for a fixed number of iterations. We select the best full solution generated by our heuristic as the final solution. In our procedure, all partial solutions are generated. However, only the

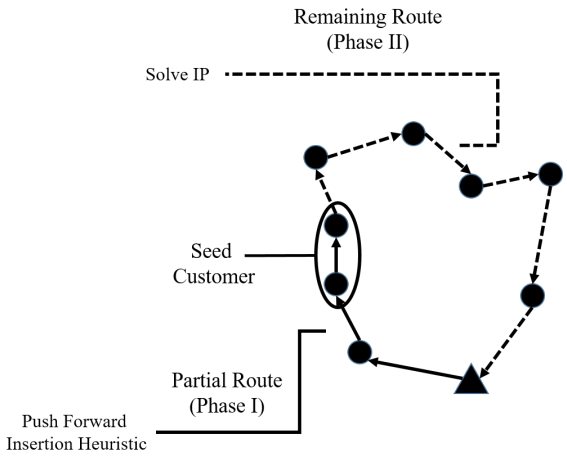


Figure 4.4: An example that shows the structure of our heuristic. The triangular node represents the depot. The circled edge is selected as the seed customer.

five partial routes with the largest value of $\frac{z(p)}{dh(p)}$ are turned into full routes, where $z(p)$ is the amount of time spent zigzagging and $dh(p)$ is the amount of time spent deadheading for partial route p. Our explanation of this restriction is given in Section 5.

4.4.1 Integer Programming Formulation

In this section, we present an integer programming formulation (IP1) for the windy rural postman problem with zigzag options. Let $G^{orig} = \{V, E\}$ be the original WRPPZTW graph. Then, we construct a graph $G^{IP} = (V^{IP}, A, M_R, A_R, M)$. V^{IP} is the set of vertices that is identical to V. A is the set of all arcs obtained by generating two arcs (i, j) and (j, i) for an edge $(i, j) \in E$. M_R is the set of arcs (i, j) and (j, i) that are generated for an edge (i, j) that you must zigzag (i.e., you only need to travel once in either direction). A_R is the set of arcs you cannot zigzag (i.e., you have to travel in required directions). M is the set of arcs where two arcs

$((i,j)$ and $(j,i))$ are generated for one zigzag optional edge (i,j) (i.e., you have to decide to travel once or twice to serve customers). We have three decision variables: (1) y_{ij} is the number of times we traverse arc (i,j), (2) x_{ij} is 1 if arc (i,j) is served with a normal traversal, 0 otherwise, and (3) zz_{ij} is 1 if edge i,j is zigzagged from i to j, 0 otherwise. For input data, c_{ij} is the traversal, or deadhead, cost of arc (i,j), s_{ij} is the service cost of arc (i,j), and m_{ij} is the extra cost if we zigzag edge (i,j) in the direction from i to j. The extra cost is calculated by $m_{ij} = z_{ij} - s_{ij}$, where z_{ij} is the zigzag cost as it was defined in Table 4.1. Note that this is different from $penalty_{ij}$ from earlier in the chapter, since m_{ij} depends on the direction the street is zigzagged, whereas $penalty_{ij}$ does not.

(IP1)

$$\text{Min} \sum_{(i,j)\in A} c_{ij}y_{ij} + \sum_{(i,j)\in M_R\cup A_R\cup M} s_{ij}x_{ij} + \sum_{(i,j)\in M_R\cup M} m_{ij}zz_{ij} \tag{4.1}$$

$$\text{s.t.} \sum_{\{(i,j)|i\in S, j\in V\setminus S\}} y_{ij} \geq 1 \qquad \forall S\subset V, 2\leq|S|\leq|V|-2 \tag{4.2}$$

$$\sum_{j\in N(i)} y_{ij} - \sum_{j\in N(i)} y_{ji} = 0 \qquad \forall i\in V \tag{4.3}$$

$$x_{ij} \leq y_{ij} \qquad \forall (i,j)\in M_R\cup M\cup A_R \tag{4.4}$$

$$x_{ij} = 1 \qquad \forall (i,j)\in A_R \tag{4.5}$$

$$zz_{ij} + zz_{ji} = 1 \qquad \forall (i,j)\in M_R \text{ and } i<j \tag{4.6}$$

$$zz_{ij} \leq x_{ij} \qquad \forall (i,j)\in M_R\cup M \tag{4.7}$$

$$zz_{ij} + zz_{ji} \leq 1 \qquad \forall (i,j)\in M \text{ and } i<j \tag{4.8}$$

$$x_{ij} + x_{ji} + zz_{ij} + zz_{ji} = 2 \qquad \forall (i,j)\in M_R\cup M, \text{ and } i<j \tag{4.9}$$

$$y_{ij} \in R^+ \qquad \forall (i,j)\in A \tag{4.10}$$

$$x_{ij} \in \{0,1\} \qquad \forall (i,j)\in M_R\cup M\cup A_R \tag{4.11}$$

$$zz_{ij} \in \{0,1\} \qquad \forall (i,j)\in M_R\cup M \tag{4.12}$$

The objective function (4.1) minimizes the total cost, which is the sum of traversal cost, service cost and zigzag cost. Constraints (4.2) ensure connectivity.

Constraints (4.3) serve as flow balance constraints, that is, whenever a vehicle visits a node, it must leave it. In constraints (4.4), for a serviced arc, the number of traversals is at least 1. Constraints (4.5) and (4.6) ensure that required arcs are traversed and zigzag required edges are zigzagged. Constraints (4.7) ensure that if we zigzag a zigzag optional edge in the direction from i to j, then we should account for the service cost as well. In constraints (4.8), if we zigzag a zigzag optional segment, then we only need to zigzag it once. In constraints (4.9), for each street that allows zigzag service, we either zigzag or serve it by traveling both directions. Constraints (4.7), (4.8), and (4.9) taken together express the concept of a zigzag optional street segment. Constraints (4.10), (4.11), and (4.12) specify nonnegativity and integer restrictions.

4.5 Computational Results

We denote our procedure by HYBRID and apply it to the most difficult test instances given by Nossack et al. [92]. Each of the 25 instances has 20 vertices, a different number of edges, and is based on a street network. For these small instances, the underlying graph is a subset of this real street network. We present results for HYBRID on successively larger instances. An instance, with 350 edges uses the entire street network that the smaller instances are based on. We used a MacBook Air (2012), running locally with 1 GB of memory allocated for the heap, and coded HYBRID in Java. Except for the IP solution, we used the architecture provided by the Open Arc Routing Library [110]. To solve the IP for the WRPPZ

Computational Results								
Instance	BNC Solution	BNC Run Time (sec)	HYBRID Solution	HYBRID Run Time (sec)	HYBRID Run Time / BNC Run Time	Lower Bound (LB)	BNC % above LB	HYBRID % Above LB
1_1_1	3065.2	43.43	3103.2	5.90	.135	3065.2	0.00	1.24
1_2_1	3231.6	1800.00	3275.6	4.65	.002	3096.2	4.37	5.79
1_3_1	3414.4	1800.00	3413.2	4.45	.002	3061.0	11.54	11.51
1_4_1	3403.4	1800.00	3412.4	6.14	.003	3089.9	10.15	10.44
1_5_1	3548.3	1800.00	3552.6	8.63	.004	3090.9	14.79	14.94
2_1_1	3067.6	0.71	3173.6	0.84	1.18	3067.6	0.00	3.46
2_2_1	3123.6	3.14	3183.2	1.82	.580	3123.6	0.00	1.91
2_3_1	3499.8	1800.00	3538.2	3.42	.002	3248.5	7.73	8.92
2_4_1	3422.4	1800.00	3454.0	3.79	.002	3240.0	5.63	6.60
2_5_1	3573.2	1800.00	3607.4	5.06	.002	3241.2	10.24	11.30
3_1_1	3483.0	6.89	3483.0	0.88	.128	3483.0	0.00	0.00
3_2_1	3677.0	1800.00	3702.4	2.60	.001	3544.3	3.74	4.46
3_3_1	3637.4	1800.00	3730.2	3.41	.002	3542.5	2.68	5.30
3_4_1	3894.8	1800.00	3921.8	6.78	.003	3598.0	8.25	9.00
3_5_1	3902.0	1800.00	3997.0	7.95	.004	3595.0	8.54	11.19
4_1_1	3067.6	0.51	3194.0	0.83	1.63	3067.6	0.00	4.12
4_2_1	3100.2	4.65	3209.6	2.60	.559	3100.2	0.00	3.52
4_3_1	3343.4	1800.00	3432.2	2.55	.001	3240.4	3.18	5.92
4_4_1	3531.6	1800.00	3565.8	5.29	.003	3226.1	9.47	10.53
4_5_1	3573.2	1800.00	3607.4	5.06	.003	3241.2	10.24	11.30
5_1_1	2806.8	89.05	2813.0	1.69	.019	2806.8	0.00	0.22
5_2_1	2854.8	1800.00	2886.8	3.40	.002	2829.1	0.91	2.04
5_3_1	3032.8	1800.00	2979.8	4.30	.002	2820.5	7.53	5.65
5_4_1	2966.2	1800.00	3042.4	5.07	.003	2831.2	4.77	7.46
5_5_1	3108.8	1800.00	3108.8	5.86	.003	2820.6	10.22	10.22
Average					0.171		5.36	6.68

Table 4.2: Comparing solutions produced by BNC and HYBRID.

(without time windows), we used CPLEX 12.6.0.

We focus on test instances with relatively tight time windows. In practice, the zigzag portion of the planning horizon is small, so these instances reflect this situation. Furthermore, the exact approach for the WRPPZTW proposed by Nossack et al. [92] (denoted by BNC) struggles on these instances. Therefore, HYBRID may be useful in solving these types of problems.

In Table 4.2, we present a computational comparison between HYBRID and BNC. Each row corresponds to a single test instance. The 25 test instances are

taken from Nossack et al. [92]. Each instance has 20 nodes. The number of edges varies in each instance. For each instance that BNC ran for 1800 seconds, the objective value corresponds to the best solution found by the exact approach. Objective values are given in columns 2 and 4, with the corresponding run times (in seconds) in columns 3 and 5 for HYBRID and BNC, respectively. Objective values in columns 2 and 4 are raw values. Times given in columns 3 and 5 are in seconds. In column 6, we give the percentage deviation of HYBRID from the BNC solution, $100(\frac{obj(HYBRID) - obj(BNC)}{obj(BNC)})\%$. Column 7 lists the percentage of time HYBRID took relative to BNC, $100(\frac{t_{HYBRID}}{t_{BNC}})\%$. We also develop a lower bound for each instance by solving a test instance with no time restrictions on the zigzag service options using our IP formulation. A second lower bound for each instance is given by the dual lower bound produced by the BNC procedure. We use the tighter of these two bounds in Table 4.2. These instances have tight time windows, so it is unlikely that these lower bounds are very tight. In columns 7, 8, and 9 of Table 4.2, we present our results. On average, HYBRID produces solutions with objective values that are 6.70% above the lower bound.

On average, HYBRID produces solutions with objective values that are 1.16% greater than the BNC solutions with the largest deviation (4.12%) on test instance 4_1_1. HYBRID produces better solutions than BNC on three test instances. HYBRID takes less than 10 seconds on all 25 test instances. BNC is able to solve only seven test instances in less than 30 minutes. On two instances, HYBRID is slower than BNC (both methods produce solutions in less than one second).

In Table 4.3, we present run times for larger test instances that we generated to demonstrate scalability. We generate 10 instances for 12 different $n \times n$ grids, where $n = 5, 6, 7, 8, 9, 10, 20, 21, 22, 23, 24, 25$. These test instances have a square grid street network, where the traversal and service costs are randomly generated between 1 and 35 independently, with uniform probability. Each street requires service with probability 0.5. The required streets can be zigzagged with probability 0.5. The streets that can be zigzagged are assigned a time window with probability 0.75. The end of the time window is 350 (or 10 times the maximum traversal cost) for the 10×10 and smaller instances. The end of the time window is 700 (or 20 times the maximum traversal cost) for the 20×20 and larger instances. The end times were chosen to ensure that the enough of the graph is still reachable within the zigzag horizon. Zigzag costs are 1.3 times the sum of the service costs. Each row in Table 4.3 contains values that are averages over the 10 instances. The row in italics shows results for the full data set (which the 20 vertex test instances were based on) provided by RouteSmart Technologies, Inc. This results in 121 instances in this set. The first column contains the number of edges in the street network. The second column contains the average total time required to solve the instances. The third column contains the average number of seed customers. The fourth column contains the average time required to complete each partial solution via the solution of the IP for the WRPPZ. The fifth column shows the average deviation from the lower bound. For the full data set, HYBRID finishes in 96 seconds. Scaling to larger instances will require large run times with HYBRID. This motivates a further performance improvement for HYBRID.

Scaling Results for HYBRID				
Number of Edges	Average Solution Time (sec)	Average Number of Seed Customers	Average Time for IP Solution (sec)	% Deviation from LB
40	4.90	3.7	1.32	28.52
60	7.36	7.5	1.02	29.88
84	9.95	10.7	0.93	27.48
112	16.85	13.4	1.25	27.83
144	20.28	18.0	1.12	31.52
180	19.42	17.8	1.11	29.41
350	*96.12*	*38.2*	*2.02*	*22.33*
760	657.81	116.6	5.56	30.27
840	910.30	130.8	6.94	29.98
924	1362.10	149.1	9.28	30.08
1012	2160.94	159.5	13.79	29.17
1104	2536.84	174.9	14.67	29.58
1200	2152.13	188.5	11.31	29.66

Table 4.3: HYBRID's performance on larger test instances. Row in italics is the full data set from RouteSmart.

Initially, we generated partial solutions using every edge that can be zigzagged with a time window. We completed all of them using the IP formulation of WRPPZ and produced the route with the lowest cost. While this approach is capable of solving the full size problem in less than 100 seconds, the scaling appears to be $O(m^2)$. On the largest instances, HYBRID takes more than 40 minutes on average to complete a run. This is due to an increase in the number of seeds and in the time it takes for the IP to complete each of the routes. We investigated a variety of partial route metrics in an attempt to retain solution quality while decreasing the number of seeds (and therefore IP evaluations) that are needed. In nearly every instance, we find that the partial route that produced the lowest cost completed route scored below average on the metric $\frac{z(p)}{dh(p)}$, where $z(p)$ denotes the amount of time spent zigzagging and dh denotes the amount of time spent deadheading on

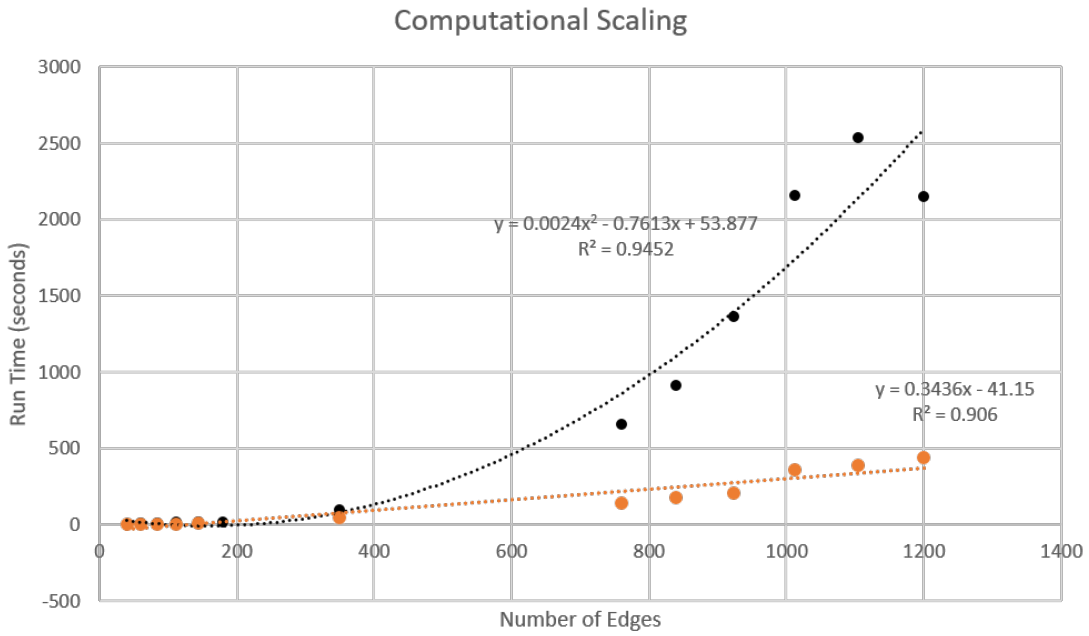


Figure 4.5: Scaling results for HYBRID (in black) and the Top Five strategy (in red) for the larger test instances.

the partial route p. We analyzed the effect of restricting our search based on this metric, (for example, completing only the five partial routes with the largest value of this ratio; we call this the Top Five strategy).

We display the results in Figure 4.5. In Figure 4.5, each point is an average of 10 instances with the same grid structure and different costs, required edges, and zigzag time windows. The black points and quadratic curve show the run times for HYBRID without the Top Five strategy. The orange points and line show the run times after applying the Top Five strategy. The equation and R^2 value for each least squares model are shown on the figure. The run time incorporates increases due to the time to solve the IP, and the number of seed solutions (each seed solution requires an IP to solve). We are able to maintain good quality solutions while

eliminating the growth of the number of seed customers as a contributing factor to the computational complexity of the heuristic. The resulting complexity is linear. A comparison between the solution quality produced by completing every route, and completing only the top five partial routes is given in Table 4.4 for the small instances.

Table 4.5 contains a comparison for the larger instances. The run times resulted from generating Figure 4.5. The right-most column shows the percent increase in objective value as a result of using the Top Five strategy. On these 121 instances (10 for each of the non-italicized rows, plus the one used to generate the small instances), the difference between the solution quality produced by the two strategies is less than 0.5% on average.

4.6 Conclusions

We presented a heuristic for solving the WRPPZTW that incorporates a new formulation for the WRPPZ capable of scaling to hundreds of nodes. We benchmarked our heuristic against exact approaches on a set of test instances. We demonstrated that our heuristic is scalable to large instances.

In future work, we want to generalize the instances to include time windows that do not necessarily start at the beginning of the planning horizon and to include time windows with different start and end times. In addition, we would like to more accurately quantify the quality of our solutions by finding a tighter lower bounding procedure and to develop a set of test instances with optimal solutions that are

Top Five Comparison			
Instance	HYBRID Deviation (%)	Top Five Deviation (%)	Difference
1_1_1	1.24	1.24	0.00
1_2_1	1.36	1.36	0.00
1_3_1	-0.04	-0.04	0.00
1_4_1	0.26	0.78	0.52
1_5_1	0.12	0.99	0.87
2_1_1	3.45	3.45	0.00
2_2_1	1.92	1.92	0.00
2_3_1	1.11	1.11	0.00
2_4_1	0.93	0.93	0.00
2_5_1	0.95	2.25	1.30
3_1_1	0.00	0.00	0.00
3_2_1	0.69	0.69	0.00
3_3_1	2.55	2.55	0.00
3_4_1	0.69	0.69	0.00
3_5_1	2.43	3.40	0.97
4_1_1	4.12	4.12	0.00
4_2_1	3.52	3.52	0.00
4_3_1	2.65	2.65	0.00
4_4_1	0.96	2.28	1.32
4_5_1	0.95	2.25	1.30
5_1_1	0.22	0.22	0.00
5_2_1	1.12	1.12	0.00
5_3_1	-1.74	-1.74	0.00
5_4_1	2.57	2.57	0.00
5_5_1	0.00	0.00	0.00
Average	1.28	1.53	0.25

Table 4.4: Comparing HYBRID with full solution for every seed customer to HYBRID with completing only top five partial routes on the small test instances.

known a priori.

Scaling Results for Top Five Strategy			
Number of Edges	HYBRID Solution Time (sec)	Top Five Solution Time (sec)	Increase in Objective Value (%)
40	4.90	2.80	0.00
60	7.36	4.13	0.87
84	9.95	5.88	0.50
112	16.85	6.39	0.47
144	20.28	8.45	0.82
180	19.42	8.23	0.68
350	*96.12*	*43.80*	*0.25*
760	657.81	143.27	0.38
840	910.30	179.42	0.30
924	1362.10	205.22	0.3
1012	2160.94	358.10	0.36
1104	2536.84	388.69	0.34
1200	2152.13	438.30	0.33
Average			0.45

Table 4.5: HYBRID's performance with and without the Top Five strategy on larger test instances. Row in italics is the full data set from RouteSmart Inc.

Chapter 5: Partitioning a Street Network into Compact, Balanced, and Visually Appealing Routes

5.1 Introduction

In routing vehicles, one dimension that has received little attention in the literature is the aesthetic quality of a set of routes. Several practitioners [111] have pointed out that the visual appeal of a proposed set of routes can have a strong influence on the willingness of a client to accept or reject a specific routing plan. In Figure 5.1, we show examples of visually appealing and unappealing routes.

It is difficult to investigate visual preferences in a quantitatively rigorous manner, given the subjective nature of personal preferences. However, there are two reasons why we believe the investigation is a worthwhile pursuit. First, it is likely that visual preferences provide intuition into how operational complexities in producing the routes are addressed in practice. For example, routes in which each driver has one geographically distinct area of responsibility can be desirable in those situations where the routing plan is followed periodically. This type of plan ensures driver familiarity with the neighborhoods that are served. Last-minute adjustments to a route can be made without significantly disturbing the routes of other drivers. Second, the visual preferences can serve as a source for new metrics that can help quantify the visual appeal of a route more precisely [112–117]. These new metrics

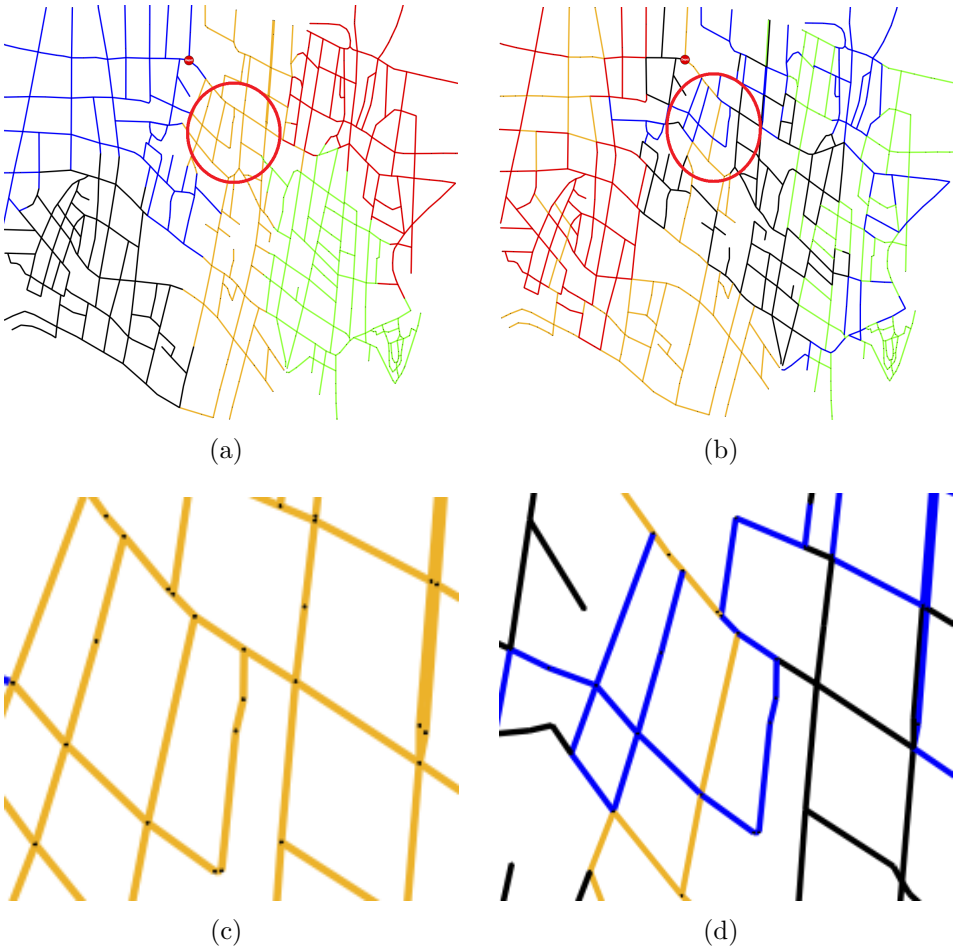


Figure 5.1: The network in (a) has visually appealing routes in which regions of service are compact and separated. Edge color corresponds to a specific route. We see that edges with the same color are clustered together and do not overlap. A set of visually unappealing routes is shown in (b). The routes overlap significantly and each visits vertices and edges that are spread throughout the graph. Panels (c) and (d) are zoomed in views of the circled area in (a) and (b), respectively.

may be used as alternatives to, or penalties in, the overall objective function.

In this chapter, we consider the Min-Max K Windy Rural Postman Problem (MMKWRPP). In the MMKWRPP, a homogeneous, fixed number of uncapacitated vehicles must traverse a predefined set of required edges in a graph with asymmetric travel costs. We select three properties of visually appealing routes (compactness, contiguity, geographic separation) and develop a new metric that takes all three

into account. We develop a cluster-first, route-second heuristic for the MMKWRPP. Computational results show that our heuristic performs favorably with respect to all of the visual metrics.

5.2 Problem Description and Terminology

The MMKWRPP has an underlying graph $G = (V, E)$ where V denotes the set of vertices and E is the set of edges. The graph is windy because its edges are undirected and the traversal costs are asymmetric. An edge is denoted by $e = (i, j)$, where $i, j \in V$ and has cost c_{ij}. This asymmetry models problems in which traversal in one direction may be much more difficult (e.g., rush hour traffic or plowing inclined streets [118]). We denote the set of edges that require service as $E_R \subseteq E$. A path through the graph is an ordered list of edges $\{e_{ij}, e_{jk}, e_{kl}\}$. The path is a cycle if it begins and ends at the same vertex. A problem instance must specify the number of vehicles available in the fleet (denoted by K).

A feasible solution to the MMKWRPP is a set of cycles (routes) $R = \{r_1, r_2, ...\}$, where each route begins and ends at a designated vertex (depot) such that there are not more than K routes ($|R| \leq K$). Each edge that requires service must be traversed by at least one vehicle ($E_R \subseteq \cup_{r \in R}$ edges(r), where edges(r) denotes the set of edges traversed in r). The objective is to minimize $z = \max_{r \in R}$ cost(r) where cost(r) is the total cost of route r. An example is shown in Figure 5.2.

Windy graphs can model undirected, directed, and mixed graphs as special cases because an arc (i, j) with cost c^a_{ij} can be represented as an edge (i, j) with $c_{ij} = c^a_{ij}$ and $c_{ji} > M$ where M is arbitrarily large. This allows solution techniques for arc

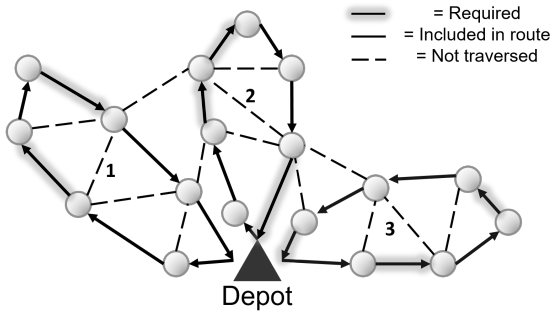


Figure 5.2: An MMKWRPP instance and a feasible solution. Highlighted edges require service. Solid edges are traversed in the solution by one of the vehicles. Dashed edges are available for traversal but are never used in this solution. The three numbered loops form the three routes in this solution.

routing problems on windy graphs to be generalized to problems with other types of underlying networks. As shown in Figure 5.3, the MMKWRPP is a generalization of many arc routing problems (e.g., the Chinese Postman Problem, the Rural Postman Problem, and min-max K variants with undirected, directed, mixed, and windy graphs). The MMKWRPP does not contain any application-specific constraints (e.g., time windows, turn penalties, etc.). Therefore, the problem serves as a basis for more specialized variants.

5.3 Literature Review

The MMKWRPP was introduced by Benavent et al. [119] in 2009. Many of the solution techniques proposed by the authors in [120, 121] for the MMKWRPP are derived from insights gained in solving the Windy Rural Postman Problem (WRPP). An integer programming formulation for the MMKWRPP, several families of valid inequalities, and an exact cutting plane procedure capable of solving small- to medium-size instances (up to 196 vertices) are given in [94]. In addition,

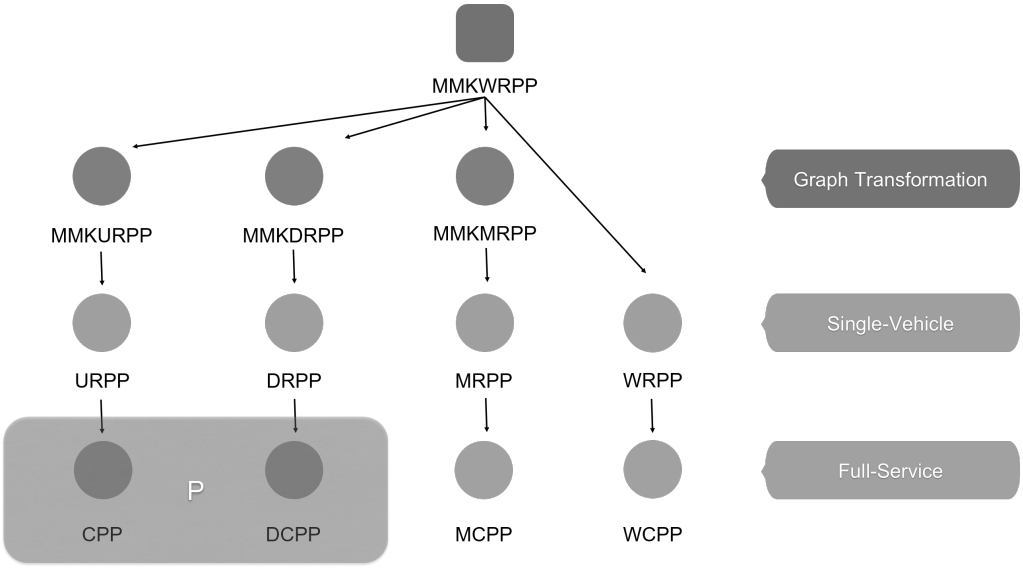


Figure 5.3: Arc routing problems that are special cases of the MMKWRPP. Windy graphs are capable of modeling undirected, directed, and mixed graphs as special cases because an arc (i, j) with cost c^a_{ij} can be modeled as an edge (i, j) with $c_{ij} = c^a_{ij}$ and $c_{ji} > M$ where M is arbitrarily large. The single-vehicle variants have $K = 1$. In the Chinese Postman variants, all edges require service. Only the directed and undirected Chinese Postman Problems are solvable in polynomial time.

the authors present three construction heuristics. Several construction heuristics are proposed, along with multi-start and scatter search algorithms in [122]. These procedures transform a WRPP into a Windy Postman Problem (WPP) which is then solved using a matching approach similar to the approach in [123]. These procedures are tested on large problem instances (up to 1,000 vertices). A compact mechanism for representing routes that is particularly convenient for improvement procedures is also given in [122]. A route is represented as an ordered list of the required edges, where it is assumed that shortest paths connect any gaps in the route. In the windy case, a shortest path problem can be solved over an auxiliary graph to determine the shortest traversal direction for a specific ordering.

The min-max objective function was first applied to arc routing problems

in [124], where a heuristic was developed for the Min-Max K Chinese Postman Problem. This problem was shown to be NP-Hard, and worst-case bounds were given for the heuristic's performance. The K Chinese Postman Problem with a min-sum objective was shown to be solvable in polynomial time [125]. Two heuristic procedures, as well as computational results are given in [126].

Most relevant to our work in this chapter is the metaheuristic procedure for the MMKWRPP in [121]. The authors present a route-first, cluster-second approach that uses the multi-start procedure from [122]. This approach produces initial solutions to the single-vehicle problem that are then split into K distinct routes and improved. In this splitting procedure, a K-arcs narrowest path (a path with at most K arcs for which the cost of the largest arc in the path is minimized) is found through a directed acyclic graph in which the vertices represent the edges in the original graph that require service (ordered by their appearance in the single-vehicle solutions). Each arc (i, j) in this path corresponds to a route in the original graph that services the required edges $i + 1, i + 2, ...j$, (in that order) and takes shortest paths between them. In Figure 5.4, we show the auxiliary graph where this path calculation is performed. In [127], a similar split procedure in a vehicle routing context is used to assess the fitness of candidates in a genetic algorithm.

A suite of intra-route improvement procedures from [122] is then applied to the individual routes, followed by a set of inter-route improvement procedures. The resulting solution is perturbed and a variable neighborhood descent is repeated a fixed number of times. This is done for a fixed number of solutions (produced by the multi-start algorithm). The global best solution is returned as the approximate

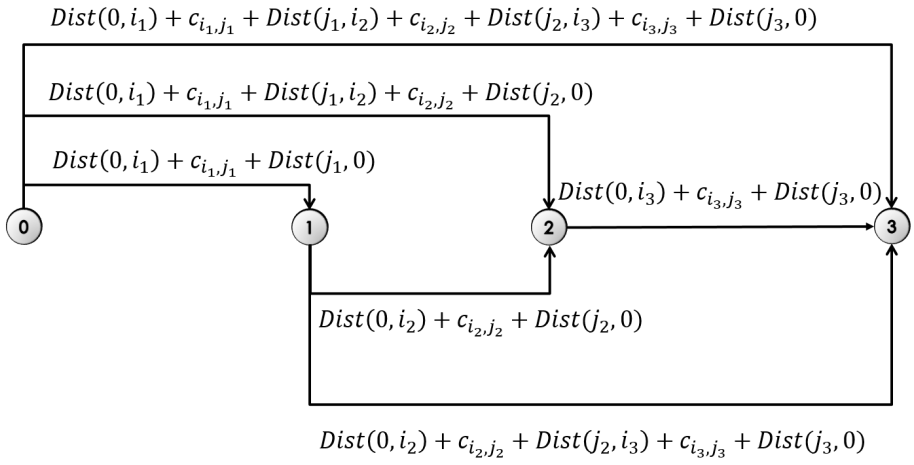


Figure 5.4: An auxiliary directed acyclic graph for an ordering of the required edges in a graph with three required edges. Let the ordering be given by e_1, e_2, e_3. Then, the costs of the arcs are given by the expressions adjacent to the edges. Vertex 0 is the depot and i_n, j_n are the endpoints of the n^{th} required edge in the ordering. Dist(i,j) is the shortest path distance between vertex i and vertex j and $c_{i,j}$ is the cost of edge (i,j). Both are calculated in the original graph.

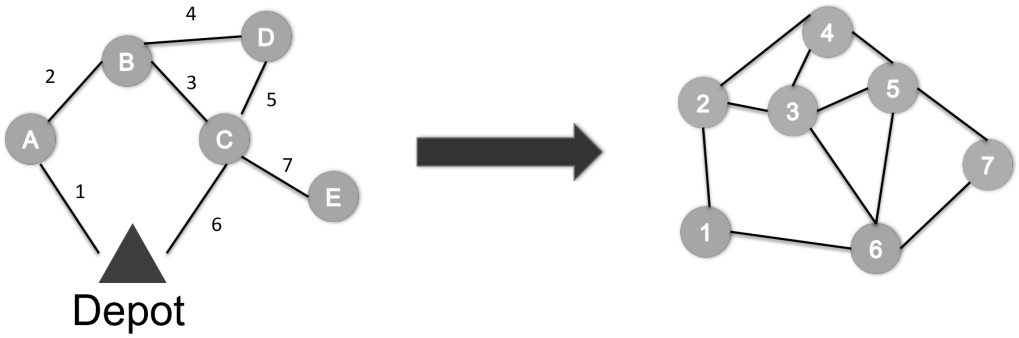


Figure 5.5: A graph and its edge-to-vertex dual. The numbers on the edges on the left correspond to the vertex numbers on the right.

solution.

5.4 Partitioning Heuristic

We develop a cluster-first, route-second heuristic for the MMKWRPP that produces routes that are compact and separated. A set of routes is compact if each driver only services customers in a small region of the graph. A set of routes is

separated if each driver services distinct regions of the graph. In the first stage of our heuristic, a vertex partitioning algorithm is adapted to partition an edge-weighted graph into K distinct regions that have approximately equal traversal cost. The partitions located farther away from the depot are adjusted to be smaller than those closer to the depot. In the second stage, a separate route is constructed to service all required edges in each partition. Finally, an improvement procedure is applied to the routes.

Given a windy graph $G = (V, E)$ (with $E_R \subseteq E$ denoting the set of required edges) and a fixed number of vehicles K available, we construct the edge-to-vertex dual graph $G^d = (V^d, E^d)$. G^d has a vertex v_{ij}^d for each edge (i, j) in E. Two vertices in G^d are connected if and only if the edges they represent share an endpoint in G. This construction is illustrated for a simple graph in Figure 5.5. Next, we seek to partition the vertices of G^d in such a way that a least-cost cycle through the required edges in each partition has roughly the same cost as the other partitions. We use the following cost function for the partitioning scheme.

$$b_{ij} = \begin{cases} 1 & \text{if } e_{ij} \text{ is in } E_{multi} \subset E, \text{ the set of edges identified by the deadhead subroutine.} \\ 0 & \text{otherwise} \end{cases} \tag{5.1}$$

$$c_{ij}^{new} = \begin{cases} (1 + b_{ij}) * c_{ij}^{orig} + \alpha * \dfrac{2 * \min\{\text{Dist}(0, i), \text{Dist}(0, j)\}}{\dfrac{|E_R|}{K}} + \text{minReqDist}(e_{ij}) & \forall e_{ij} \in E_R \\ 0 & \text{otherwise} \end{cases} \tag{5.2}$$

where α is a parameter that affects the distance penalty relative to the original edge cost, and $\text{minReqDist}(e_{ij})$ is a function that generates the shortest distance from e_{ij} to any other edge in E_R.

Before describing (5.1) and (5.2) in detail, it is helpful to consider the weakness of assigning cost e_{ij} in the original graph to vertex v_{ij} in G^d. In this assignment, the sum of the edge costs in each partition would be approximately equal. However, for partitions far from the depot, there is also the cost that is incurred to travel to and from the service area. Therefore, partitions far from the depot will take longer to service than those close to the depot. To achieve route balance, we would, therefore, like partitions farther away from the depot to have fewer customers. In addition, this cost function does not distinguish between customers and edges that do not require service. While the sum of the edge costs may be the same, a partition with few customers can be serviced quickly, and many of the streets may never be traversed. A more sophisticated cost function would only consider the costs of the streets that are likely to be included in the routes that service the partitions.

We now describe each of the terms in our modified cost function. The first term in c_{ij}^{new} penalizes edges in the graph that will have to be traversed multiple times (deadheaded) in the final solution, where c_{ij}^{orig} is the original cost of e_{ij} in G. The deadhead algorithm is a preprocessor that identifies edges in the final solution that will have to be traversed multiple times. The procedure looks for vertices of degree 1, removes them iteratively, and adds each removed edge to the set produced by the subroutine. An outline of this procedure is given in Appendix B. The second term in c_{ij}^{new} penalizes each partition by the distance to the depot. The intuition is that $|E_R|/K$ estimates the number of edges in each partition. The factor of two incorporates the path to and from the partition. If the number of required edges in each partition is nearly equal, a value of α near 1 should produce good results.

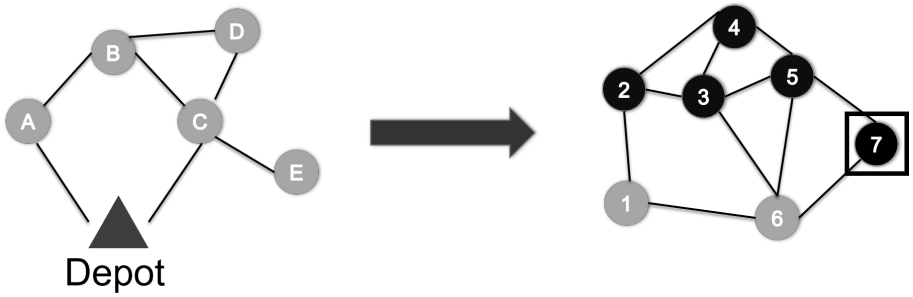


Figure 5.6: Cost penalties applied prior to running the partitioning algorithm. All vertices have the penalty associated with the minReqDist function, so this penalty is not represented in the figure. The black vertices have an additional cost penalty proportional to min $\{\text{Dist}(0,i),\ \text{Dist}(0,j)\}$ where the corresponding edge in the original graph is (i,j). Vertex 7 is both shaded black and surrounded by a square to reflect that it has an additional multiple of the edge cost since we know that we are going to have to deadhead it.

Finally, the last term in c_{ij}^{new} penalizes each required edge by the minimum distance required to reach another required edge (or the depot). After servicing a street, a vehicle will need to travel at least this distance to continue the route. In Figure 5.6, we show the penalties applied to the vertices building on the example given in Figure 5.5.

Using these costs, we partition G^d via the open source METIS partitioning library [128]. We use the gpmetis routine from the library. This algorithm coarsens a graph iteratively to reduce the complexity of the partitioning problem by collapsing connected subsets of the vertex set. Once the graph is easy to partition (in our case, once it has only K vertices), it is then relaxed (iteratively re-expanded) and, at each step, the partition is refined to achieve a balanced result [129].

This partition corresponds exactly to a labeling of the required edges in the original graph G, where each receives a label from the set $P = 1, 2, ..., K$. Let G_p be the graph identical to G where only the required edges with label $p \in P$ are

required. For example, G_1 will have the same number of vertices and connections as in G, but only the edges in the first partition are required. This correspondence is shown for a small example in Figures 5.7 and 5.8. Then, a heuristic procedure for the WRPP [122] solves the single-vehicle problem on each of the G_p and these form the routes in the initial feasible solution. After an initial solution is generated, a single pass of the improvement procedures from [121] can improve the solution. These improvement procedures consider a set of customer swaps from one route to another. In [121], these procedures are only guaranteed to improve performance with respect to the objective function. Performance can degrade with respect to the aesthetic metrics. However, we modify these procedures so that moves that degrade the route overlap index (defined in (5.6)) are not considered. This does not have a large effect on the objective function value (a less than 1% increase, on average). The local search procedures are applied one time, as we do not perturb the solution. Multiple initial solutions are created by varying the value of α.

5.5 Compactness and Separation of Routes

The aesthetic characteristics of a feasible solution are important because compact, distinct, continuous routes allow drivers to become familiar with a particular neighborhood [117]. A hybrid clustering and greedy-insertion initialization procedure is developed in [116] to address the objective of maximizing the aesthetic attractiveness of routes for a commercial waste management company. Solution techniques that failed to produce appealing routes were rejected by a Dutch transportation consulting company in [115]. A clustering approach to the VRP with time

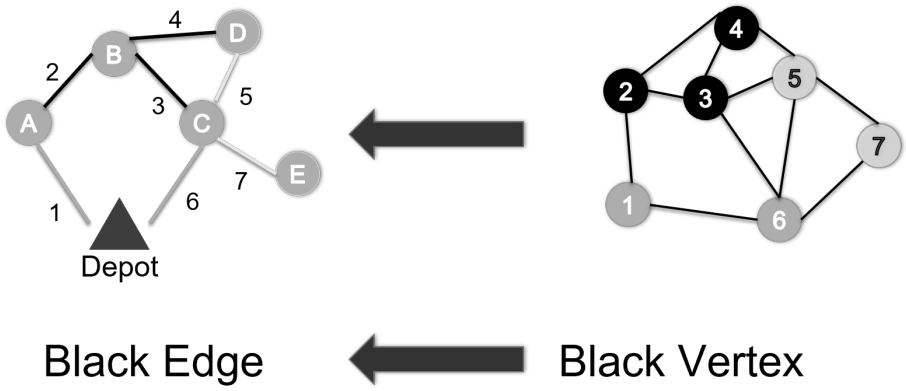


Figure 5.7: A vertex partition of the edge-to-vertex dual, and the corresponding edge partition of the original graph G. Vertices 1 and 6 are in one partition, vertices 2, 3, and 4 are in a second partition, and vertices 5 and 7 are in a third partition.

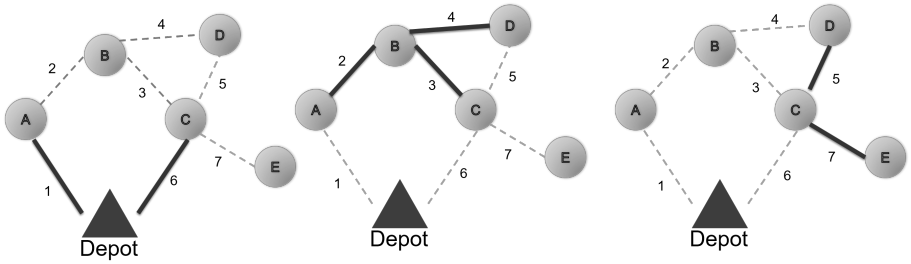


Figure 5.8: Given the partitioning on the left of Figure 5.7, one graph for each of the partitions is created in which only the edges in the partition are required, and all others are marked as unrequired. A single-vehicle WRPP solver is used on each graph to create a route in the solution to the K-vehicle problem.

windows is proposed in [130] to produce compact routes while generating competitive results with respect to the objective function, although no attempt is made to quantify the compactness of the resulting routes. Similarly, a K-means approach is developed in [131] to partition customers in large-scale VRP instances.

This additional requirement has led to the development of several measures that attempt to formally capture the visual quality of a routing plan. He et al. [131] use N_h, the number of vertices that belong to more than one route's convex hull. Lu and Dessouky [113] propose the crossing length percentage metric. Crossing

points within the context of a single route are identified. The ratio of the route that is segmented off by this crossing (summed over all crossing points) is computed. The authors use this metric to guide an insertion procedure to improve the visual quality of each route. Poot et al. [115] provide several candidate metrics including the average number of vertices closer to the center of a route they are not on, the number of vertices contained in a different route's convex hull, and the number of crossings in a tour. Although these measures evaluate candidate solutions, they are not considered or incorporated during the construction of the routes. Tang and Miller-Hooks [117] introduce the median of a tour (the vertex that minimizes the maximum distance to any other vertex on the tour) and give several metrics for compactness and overlap. They propose the total number of vertices closer to another route's median (similar to Poot's measure, but substituting the location of the median for the geographic center). The authors show that if every vertex is closer to its route's median, then their convex hulls will not overlap. They also propose the average Euclidean distance between a route's median and its customers. The authors design a procedure around manually chosen medians, arguing that the use of human computer interaction can improve solution quality, and avoid unacceptable routes. Matis [114] presents three more metrics: one to measure compactness (COMP), one to measure overlap (DGRB), and a combined, normalized metric of visual attractiveness (VA) given in (5.3), (5.4), and (5.5).

$$\text{COMP} = \frac{\text{AvgDistance}}{\text{AvgMaxDistance}} \tag{5.3}$$

$$\text{DGRB} = 2 * \left(1 - \frac{|\hat{O}|}{|O|}\right) - 1 \tag{5.4}$$

$$\text{VA} = \frac{1}{\frac{\text{NC}}{5} + \frac{1}{\text{COMP}} + \frac{1}{\text{DGRB}} - 1} \tag{5.5}$$

AvgDistance and AvgMaxDistance are the average distance between consecutive customers, and the average distance restricted to the greatest 20% of consecutive distances, respectively. $|\hat{O}|$ and $|O|$ are the number of customers closer to another route's center (of gravity) and the number of customers on the route, respectively. NC is the number of crossings within a route. The value of VA can be 1; in practice, its value is usually less than 0.5.

The first set of multi-route metrics were proposed by Constantino et al. in [112]. The authors give three metrics.

$$\text{Route Overlap Index (ROI)} = \frac{\text{NO} - |N|}{(\sqrt{|R|} + \sqrt{|N|} - 1)^2 - |N|} \tag{5.6}$$

$$\text{Average Traversal Distance (ATD)} = \frac{\sum_{r=1}^{|R|} \sum_{a,b \text{ served by } R} \text{Dist}_{ab}}{|\text{taskpairs}|} \tag{5.7}$$

$$\text{Connectivity Index (CI)} = \frac{CC}{|R|} \tag{5.8}$$

$|R|$ is the number of routes. N is the number of vertices in the graph, including the

depot. (Since we do not perform graph simplifications that remove vertices with no incident, required edges, we use N to denote the number of vertices with an incident required edge). NO $= \sum_{i \in N} \sum_{r=1}^{|R|} n_i^r$ is the number of routes a vertex belongs to, summed over all vertices that have an incident required edge. |taskpairs| $= \frac{|\text{tasks}| * (|\text{tasks}| - |R|)}{2|R|^2}$ is the approximate average number of tasks assigned to each route. CC is the number of connected components of the required edges in the candidate solution. These three metrics try to capture the separation, compactness, and contiguity of the routes (where smaller values of the metrics correspond to distinct, compact routes). The authors present a hybrid IP-heuristic method that uses the NO metric as part of the objective function.

The literature on political districting provides several compactness metrics that try to automatically detect gerrymandering. The Reock metric and Schwartzberg metric are widely used [132]. The Reock metric (5.9) calculates the ratio of the area enclosed by the perimeter of a partition to the area of the smallest circle that includes that area. The Schwartzberg metric (5.10) calculates the ratio of the perimeter length to the circumference of the circle with the same area.

$$\text{Reock} = \frac{\text{A(D)}}{\pi r_{\text{enc}}^2} \tag{5.9}$$

$$\text{Schwartzberg} = \frac{\text{P(D)}}{2\sqrt{\pi \text{A(D)}}} \tag{5.10}$$

In (5.9) and (5.10), D is the polygon defined by the perimeter of the district or partition under consideration. A(D) is the area of D, r_{enc} is the radius of the smallest circle enclosing D, and P(D) is the length of the perimeter of D. Larger values of the Reock metric and smaller values of the Schwartzberg metric correspond to compact districts.

We propose a metric for a graph that lies (approximately) in the coordinate plane. Our metric tries to address several weaknesses of the other multi-route metrics. Our metric is defined as follows. For each route r in R and each vertex v_i^r in the set of vertices V^r visited by route r, let (x_i, y_i) be the coordinates of v_i. Let convex(r) be the convex hull of the vertices in V^r and intersec(H_1, H_2) be the area of the intersection between hulls H_1 and H_2. We call our metric hull overlap (HO).

$$\text{HO} = \sum_{r_1, r_2 \in R} \left(\frac{\text{intersec}(\text{convex}(r_1), \text{convex}(r_2))}{\text{convex}(r_1)}\right) \frac{1}{|R|} \tag{5.11}$$

HO captures the geographic overlap between the areas of responsibility of the drivers. We are motivated by [117] and note that if every endpoint of a required edge is closest to its own route's median, then the convex hulls of the routes will not overlap. Unlike ROI, HO penalizes out-of-place edges the more that they are out-of-place relative to ideal boundaries. For example, consider a postman who services customers in the west part of town, except for one customer serviced in the east part of town. Clearly, this is a more egregious violation than one customer in the

central part of town. HO combines the desire for non-overlapping and contiguous routes. Our metric does not consider shape. While ATD penalizes oblong routes more than square or circular boundaries, HO will not. It is not clear that this sort of bias is inherent in the compactness that ATD is trying to capture. In Figure 5.9, we show hull overlap calculations for a small instance.

We note that there are practical considerations and street network topologies that can be problematic for all of these metrics. For example, choke points (e.g., bridges and tunnels) can cause ROI to label all reasonable sets of routes as having poor visual quality. In addition, streets that intersect in the plane, but do not meet at an intersection, can reduce the usefulness of the ROI metric. Obstructions (e.g., rivers and walls) and variable street density may also cause the best routes (with respect to the min-max objective and visual appeal) to have significantly overlapping convex hulls. Figure 5.10 depicts these scenarios. These considerations can make it difficult to compare solutions for different instances without some additional normalization.

However, we believe that these are minor weaknesses that do not degrade the utility of the metrics for comparing sets of routes for a fixed instance. We also did not observe these problematic network topologies in any of the test instances, so we believe it is rare for them to occur in practice. As we shall see in the next section, the metrics are usually in agreement, which would not be the case if some of the problematic conditions were present. We advocate using a suite of metrics so that even when one metric does provide an incorrect assessment, the overall assessment will still likely be correct.

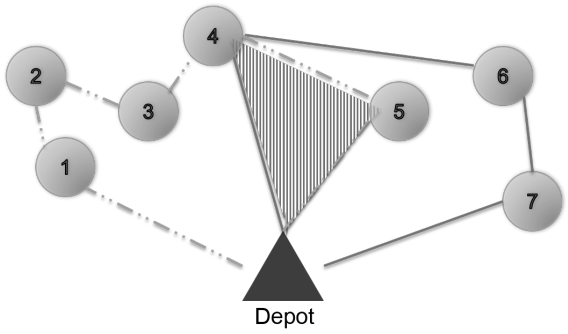
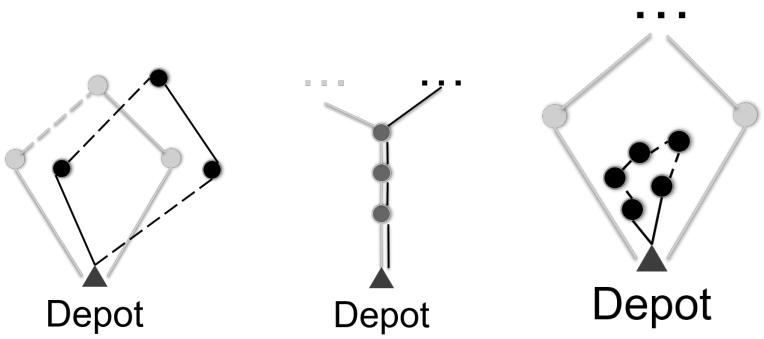


Figure 5.9: Hull overlap calculations for a small instance. The two routes are r_1 =(Depot, 1, 2, 3, 4, 5, Depot) and r_2 =(Depot, 4, 6, 7, Depot). The striped region is the overlap of their two convex hulls. If the overlap was 20% of r_1's convex hull and 30% of r_2's convex hull, then the value of HO for this set of routes would be $\frac{.2+.3}{2} = .25$.



(a) (b) (c)

Figure 5.10: Three scenarios that cause ROI and HO to misclassify the visual quality of the routes. Edges and vertices are shaded according to the route they are on, so one vehicle traverses the gray edges and visits the gray vertices, while the other visits the black edges and black vertices. If a vertex is visited by multiple vehicles, the shading is arbitrary. Dotted lines are non-required streets and solid lines are customers. The ellipsis symbol (...) indicates that an unspecified segment of the route is located at this position. In (a), ROI will identify this set of routes as non-overlapping. In (b), ROI will identify this set of routes as overlapping. While they do overlap, this degree of overlap is inevitable in any solution. In (c), HO will identify this set of routes as overlapping and visually unappealing when this may be the visually ideal solution.

5.6 Computational Results

We present results for two classes of test instances. The first set is based on real-world street networks retrieved from the Open Street Maps project [15]. In these instances, the traversal costs are based on actual distances. Each edge is given an initial cost of the length in meters, and then each direction is assigned a cost randomly and independently between 95% and 105% of the initial cost (from a uniform distribution). One example, based on the street network of Istanbul, Turkey was shown in Figure 5.1, with our heuristic producing the visually appealing solution and the route-first heuristic producing the alternative solution. There are eight real-world street networks. Each is named for the city where the network is located. The second set contains artificially created rectangular instances of various sizes whose edges have costs uniformly distributed between 1 and 10 (similar to the first set, each direction's cost was determined independently). We generate and test 10 artificial, rectangular instances denoted by R1, R2, ..., R10. In Figure 5.11, we show the routes for R1 produced by our partitioning heuristic (denoted by PAR) and Benavent et al.'s route-first heuristic (denoted by RF). We create a set of large test instances using the graph generators in the Open Arc Routing Library [110]. We applied the metaheuristic in [121] and our partitioning heuristic to both sets of test instances. We report solution quality and results for the ROI and ATD metrics from [112] as well as our HO metric. It is important to note that there are two parameters in the procedure in [121] whose values need to be set: (1) number of initial solutions produced and (2) number of perturbations per solution that are

performed. Several combinations are tested in [121]. During our computational experiments, these proposed values led to very long running times for our set of test instances. In practice, perturbing the solution rarely managed to improve the current best solution. This is expected because the number of recommended perturbations represented a much larger portion of the search space on the set of smaller test instances used by the authors in [121]. With this in mind, we limited the search to generating 10 initial solutions. We performed 10 perturbation iterations for each solution, so that PAR and RF have approximately the same run time on both sets of instances.

Recall that PAR has a parameter α that affects the contribution of the distance term in the costs used to partition the graph. In order to understand the effect of the value α on the solution quality, PAR solved each instance 100 times. We used equally-spaced values of the coefficient of the second term in our modified cost function $(\frac{2\alpha|E_R|)}{k})$ in the interval 0 to 0.1. This interval corresponds to values of α from 0 to 5, above which the contribution from the term with α, (in (5.2)) was approximately equal to the sum of the costs of the edges in the network. For larger values of α, the contribution from this penalty is greater than the sum of the costs of the edges in the network, and the balance of the actual required workload is no longer the primary contributing factor to the rebalanced costs. This is undesirable because running the heuristic with these values produces a partition that effectively ignores part of the problem structure. In Figure 5.12, we plot the objective value for the Auckland, New Zealand instance using 10 vehicles for each value of this

coefficient.

We created our cost function so that a value of $\alpha \approx 1$ should produce optimal results. However, recall that the cost function approximates the number of required edges in each partition by the average ($|E_R|/k$) even though we expect partitions farther from the depot to contain fewer edges than those close to the depot. To account for these deviations we search with equally spaced values in the interval $[.5, 1.5]$ in PAR.

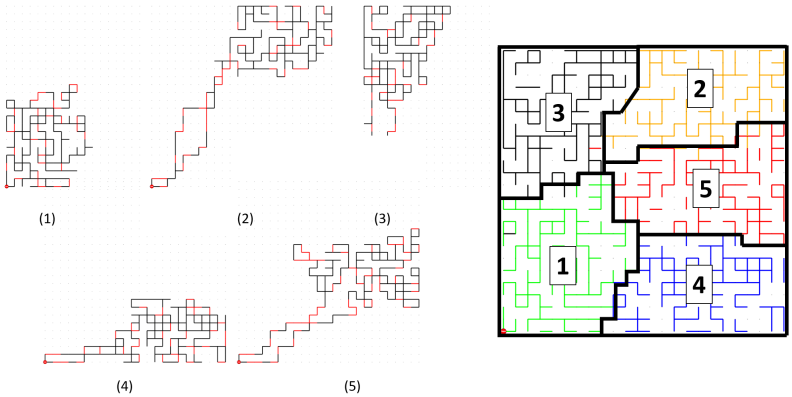
In order to validate this choice of parameter space, we used the irace software package developed in [133]. This software uses an iterated racing algorithm to produce optimal or near-optimal input values given a parameterized algorithm. While these experiments produced different values of α (roughly a value of two when trained on the rectangular instances and between four and five when trained on the real street networks), we observed only a very small improvement with respect to the objective function (on average, less than 0.1%), while the performance with respect to the aesthetic metrics degraded substantially. Therefore, we proceeded with values of α between .5 and 1.5.

For each of the 18 underlying street networks (eight real-world street networks and 10 artificial rectangular networks), we generate six problem instances for a total of 108 test instances. These instances are produced by varying the number of vehicles available in the fleet (3, 5, 10) and the location of the depot (near the periphery or the vertex closest to the geographic center of the network). In Tables 5.1 - 5.6, we present results produced by RF and PAR on these instances.

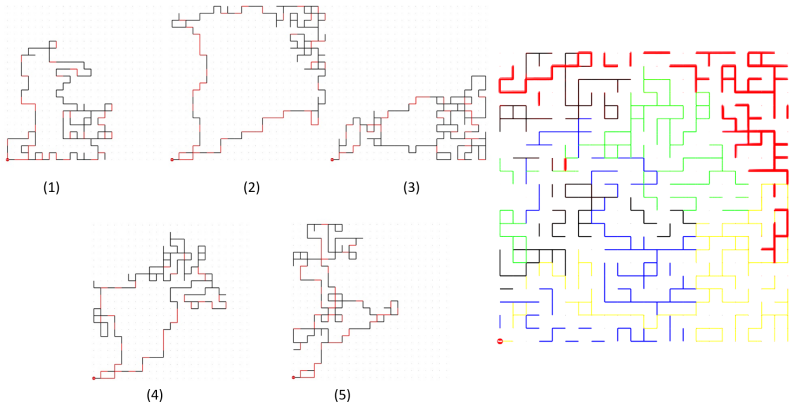
Run times for all six categories of test instances were comparable using a

Macbook Air (2012) with a 1.8GHz dual-core Intel Core i5 processor. For the 10, five, and three vehicle instances, PAR had an average run time of 665, 543, and 507 seconds, respectively. RF had an average run time of 653, 472, and 432 seconds, respectively. We selected a set of values for α that keeps the run times for PAR and RF nearly equal. During preliminary testing, increasing the number of values for α had negligible impact on solution quality.

In Tables 5.1-5.6, the columns labeled n and m give the number of vertices and edges in the network, respectively. In the RF and PAR columns, we report the absolute performance for



(a) A set of five visually appealing routes produced by PAR on the R1 instance (the largest of the rectangular instances). The panels on the left show each of the routes individually, with black edges requiring service. The panel on the right shows the five routes on the same graph. The bold lines show route boundaries for customers assigned to each route.



(b) A set of five visually unappealing routes produced by RF on the R1 instance. We point out that the area of the graph serviced by (2) spans the width of the graph, and overlaps the areas serviced by the other routes. In the right panel, customers served on route (2) are shown in red.

Figure 5.11: Routes produced by PAR and RF on the R1 instance.

Benavent et al.'s heuristic [121] and our approach for the min-max objective (Objective). The scores of RF and PAR on the aesthetic metrics are given by the columns labeled ROI, ATD, and HO. Values for the aesthetic metrics are produced by (5.6), (5.7), and (5.8). Lower values indicate more visually appealing solutions. The %

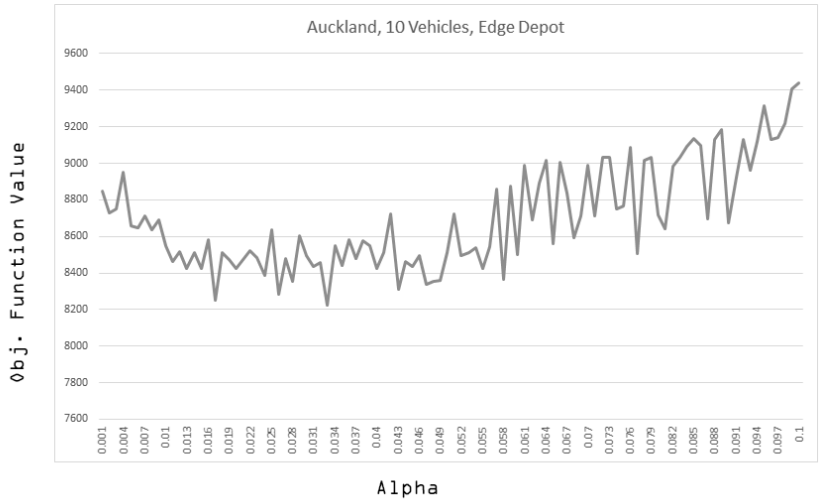


Figure 5.12: Plot of the partitioning approach's performance for different values of the coefficient of the distance term for the Auckland instance with 10 vehicles. The depot is located on the periphery of the graph. Despite the oscillations, it is clear that there is an optimal search range centered around 0.033. This general shape is typical of instances where the depot is far from the geographic center of the graph.

Deviation column gives the relative performance of the two heuristics. This percentage is generated by $100(P - R)/R$ where P is the performance of PAR and R is the performance of RF.

Each table provides results for a particular instance profile (a fixed depot location and fleet size). Over all instance profiles, PAR performs the best on the 18 instances with the smallest fleet size ($k = 3$) and with the depot located centrally.

In Table 5.1, for this profile, PAR has a deviation of less than 1% on average with respect to the objective function. PAR produces the smallest values of ROI and ATD on all 18 instances, and HO on 17 instances with RF producing the smallest value of HO on one instance (San Francisco). On average, PAR outperforms RF, by 63% on ROI, by 17% on ATD, and by 56% on HO.

On average, the results across all four metrics for the five-vehicle instances with

a central depot location (Table 5.2) are within 10% of the results for the three-vehicle instances (i.e., PAR outperforms RF by 20.8% in the five-vehicle case as opposed to 16.9% in the three-vehicle case, and similarly with the other three metrics). The solutions produced by PAR outperform RF with respect to the min-max objective, by 0.03% on average. Again, RF does better than PAR on HO for the San Francisco instance. This is due to the improvement procedures moving a few customers to routes whose region of responsibility is relatively far from the displaced customers. The difference between RF and PAR on HO is larger in the case with more vehicles because HO is measured as a percentage, and the areas of responsibility are smaller when more vehicles cover the same street network.

With the central 10-vehicle instances (Table 5.3), the effect of increasing the fleet size begins to become apparent. PAR is consistently at least 2% worse than the min-max objective performance of RF and averages 4% worse for this profile. The difference in the performance of PAR as compared to RF on HO decreases from 55.8% (with three vehicles) to 46.3% (with five vehicles) to 30.5% better (with 10 vehicles). For ATD, we observe the opposite trend, with PAR's advantage growing with fleet size from 17% (with three vehicles) to 20.8% (with five vehicles) to 27% better (with 10 vehicles). PAR outperforms RF by approximately 70% on ROI regardless of fleet size.

For those instances where the depot is located near the edge of the network (Tables 5.4, 5.5, 5.6), we see that instances with larger fleet size cause PAR to perform less favorably than RF. The longer travel distances to and from each of the partitions means that RF has more opportunities to exploit shortcuts throughout

the graph than PAR. For each additional vehicle, there is a roughly 1.1% increase (Figure 5.15) in the difference between RF and PAR with respect to the min-max objective (0.7% for three vehicles, 4.7% for five vehicles, and 9.1% for 10 vehicles, on average). Similar trends for ATD and HO hold as in the central depot case. For ATD, each additional vehicle brings a 2.6% increase in PAR's advantage while, for HO, it decreases by 3.2%. As the fleet size increases, and PAR's compactness advantage (as measured by ATD) increases, RF's advantage in geographic overlap (as measured by HO) is due to the fact that each hull is larger, decreasing the percentage of overlap. Finally, for ROI, PAR again holds a consistent advantage of 80% over RF regardless of fleet size (89.1% for three vehicles, 82.6% for five vehicles, and 80.9% for 10 vehicles).

On average, over all instances based on real street networks, RF outperforms PAR with respect to the min-max objective by 2.36%. PAR outperforms RF by 81.7% on ROI, 22.9% on ATD, and 26.8% on HO. For the rectangular instances, RF is better than PAR by 4.38% on the min-max objective. PAR is better than RF by 72.7% on ROI, by 29.6% on ATD, and

Table 5.1: Solution Quality for Centered Depot Instances with Three Vehicles

			RF				PAR				% Deviation			
Instance	n	m	**Objective**	**ROI**	**ATD**	**HO**	**Objective**	**ROI**	**ATD**	**HO**	**Objective**	**ROI**	**ATD**	**HO**
San Francisco, CA	703	840	13402	.871	1344.36	.184	13696	.174	1198.89	.189	2.2	-80.0	-10.8	3.2
Washington D.C.	593	663	11296	1.611	1553.97	.165	11694	.157	1337.22	.116	3.5	-90.1	-13.9	-29.2
London, UK	855	1004	8511	1.388	943.42	.728	8631	.261	746.84	.301	1.4	-81.1	-20.8	-58.5
Istanbul, TR	693	865	12061	2.329	878.75	.702	12074	.614	770.34	.309	0.1	-73.6	-12.3	-55.8
Perth, AU	532	592	8994	.702	1036.41	.324	9191	.167	823.43	.204	2.2	-76.1	-20.5	-36.9
Auckland, NZ	1209	1297	19817	1.456	1655.70	.408	19844	.132	1599.28	.155	0.1	-90.9	-3.4	-61.9
Helsinki, FI	1310	1540	10183	1.090	882.89	.837	10259	.235	666.47	.394	0.7	-78.4	-24.5	-52.8
Vienna, AT	506	586	4915	1.353	740.64	.521	4886	.169	579.60	.234	-0.5	-87.5	-21.7	-54.9
R1	576	1104	1237	1.451	84.46	.740	1251	.580	66.206	.243	1.1	-60.0	-21.6	-67.0
R2	529	1012	1079	1.618	79.560	.901	1086	.580	61.660	.262	.64	-64.1	-22.4	-70.9
R3	484	924	1021	1.550	68.002	.696	1001	.633	58.830	1.76	-1.9	-59.1	-13.4	-74.7
R4	441	840	949	.861	60.668	.244	954	.530	58.708	.215	.52	-38.4	-3.2	-11.7
R5	400	760	807	1.786	65.908	.872	821	.385	51.339	.227	1.7	-78.4	-22.1	-73.8
R6	361	684	737	1.033	60.214	.683	733	.442	51.541	.245	-.54	-57.1	-14.4	-64.1
R7	324	612	708	1.498	61.262	.821	708	.345	48.594	.204	0.0	-76.9	-20.6	-75.1
R8	289	544	566	1.444	51.362	.645	578	.453	44.500	.216	2.1	-68.5	-13.3	-84.6
R9	256	480	504	1.357	57.157	1.025	511	.394	41.052	.157	1.3	-70.9	-28.1	-84.6
R10	225	420	467	1.365	47.644	.736	460	.470	39.065	.231	-1.4	-65.5	-18.0	-68.6
Average											.73	-63.2	-16.9	-55.8

Table 5.2: Solution Quality for Centered Depot Instances with Five Vehicles

			RF				PAR				% Deviation			
Instance	n	m	**Objective**	**ROI**	**ATD**	**HO**	**Objective**	**ROI**	**ATD**	**HO**	**Objective**	**ROI**	**ATD**	**HO**
San Francisco, CA	703	840	8772	.936	1108.72	.354	9012	.2722	1047.97	.542	2.7	-70.9	-5.4	53.3
Washington D.C.	593	663	7770	1.535	1204.31	.626	7684	.258	1020.80	.424	-1.1	-83.1	-15.2	-32.1
London, UK	855	1004	5484	1.291	755.77	.711	5522	.215	548.23	.499	0.6	-83.3	-27.4	-29.7
Istanbul, TR	693	865	7680	2.777	822.70	1.129	7655	.495	528.67	.574	-0.3	-82.1	-35.7	-49.0
Perth, AU	532	592	6112	1.485	830.07	.706	5921	.117	767.16	.488	-3.1	-92.1	-7.5	-30.8
Auckland, NZ	1209	1297	12711	1.879	1548.49	.911	12629	.142	1305.25	.223	-0.6	-92.4	-15.7	-75.4
Helsinki, FI	1310	1540	6373	1.419	677.10	.790	6467	.238	627.51	.758	1.4	-83.1	-7.3	-4.0
Vienna, AT	506	586	3362	1.741	597.24	.924	3171	.138	429.64	.353	-5.7	-92.0	-28.0	-61.7
R1	576	1104	767	1.190	62.707	1.028	777	.595	50.409	.357	1.3	-50	-19.6	-65.2
R2	529	1012	677	1.699	68.423	1.155	674	.429	45.885	.328	-.43	-74.7	-32.9	-71.5
R3	484	924	636	1.482	54.942	.940	639	.389	43.717	.347	.47	-73.5	-20.4	-63.0
R4	441	840	595	1.512	54.402	.868	608	.600	45.135	.421	2.2	-60.2	-17.0	-51.4
R5	400	760	510	1.761	51.830	1.091	523	.491	39.548	.371	2.5	-72.0	-23.6	-65.9
R6	361	684	463	1.358	50.857	1.149	471	.388	38.288	.340	1.7	-71.4	-24.8	-70.3
R7	324	612	446	1.099	45.943	.604	452	.718	37.698	.437	1.3	-34.6	-17.9	-27.6
R8	289	544	372	1.805	48.718	1.270	379	.409	33.137	.420	1.8	-77.3	-31.9	-66.8
R9	256	480	326	1.199	42.596	.941	325	.561	32.308	.442	-.30	-53.1	-24.3	-52.9
R10	225	420	309	1.260	38.084	.928	293	.356	30.201	2.75	-5.1	-71.7	-20.6	-70.3
Average											-.03	-73.2	-20.8	-46.3

Table 5.3: Solution Quality for Centered Depot Instances with 10 Vehicles

			RF				PAR				% Deviation			
Instance	n	m	**Objective**	**ROI**	**ATD**	**HO**	**Objective**	**ROI**	**ATD**	**HO**	**Objective**	**ROI**	**ATD**	**HO**
San Francisco, CA	703	840	5221	1.650	996.48	1.197	5944	.200	788.78	.895	13.8	-87.8	-20.8	-25.2
Washington D.C.	593	663	6329	1.302	1132.48	1.161	6329	.186	682.37	.572	0	-85.6	-39.7	-50.6
London, UK	855	1004	3274	1.631	571.28	1.222	3352	.207	401.95	.989	2.4	-87.3	-29.6	-19.0
Istanbul, TR	693	865	4403	2.809	631.47	1.770	4577	.666	434.81	1.27	3.9	-76.2	-31.1	-28.0
Perth, AU	532	592	3600	1.497	632.78	1.035	3638	.109	471.25	1.052	1.1	-92.7	-25.5	1.6
Auckland, NZ	1209	1297	7345	2.160	1315.97	1.201	7850	.124	961.61	.846	6.8	-94.2	-26.9	-29.5
Helsinki, FI	1310	1540	3508	1.599	553.96	1.866	3625	.360	505.93	1.356	3.3	-77.4	-8.6	-27.3
Vienna, AT	506	586	1985	1.570	469.58	1.019	2094	.154	345.41	.759	5.4	-90.1	-26.4	-25.5
R1	576	1104	436	1.687	52.822	1.653	464	.524	36.452	.844	6.4	-68.9	-30.9	-48.9
R2	529	1012	373	1.392	45.977	1.459	388	.470	33.586	.931	4.0	-66.1	-26.9	-36.1
R3	484	924	353	1.555	41.231	1.232	370	.414	32.355	.918	4.8	-73.3	-21.5	-25.4
R4	441	840	334	1.496	42.678	1.114	353	.455	32.750	.910	5.6	-69.5	-23.2	-18.3
R5	400	760	290	1.624	38.634	1.385	305	.606	.28.655	.950	5.1	-62.6	-25.8	-31.4
R6	361	684	271	1.540	40.142	1.496	277	.505	28.627	.935	2.2	-67.1	-28.6	-37.4
R7	324	612	259	1.650	36.492	1.405	269	.512	26.527	.967	3.8	-68.9	-27.3	-31.1
R8	289	544	216	1.781	36.880	1.639	220	.441	24.151	.766	1.8	-75.1	-34.5	-53.2
R9	256	480	193	1.304	31.286	1.546	201	.396	22.666	1.120	4.1	-69.5	-27.5	-27.5
R10	225	420	180	1.169	31.066	1.448	188	.410	21.364	.964	4.4	-64.9	-31.2	-33.4
Average											4.4	-76.5	-27.0	-30.3

Table 5.4: Solution Quality for Edge Depot Instances with Three Vehicles

			RF				PAR				% Deviation			
Instance	n	m	**Objective**	**ROI**	**ATD**	**HO**	**Objective**	**ROI**	**ATD**	**HO**	**Objective**	**ROI**	**ATD**	**HO**
San Francisco, CA	703	840	14658	1.162	1413.46	.762	14657	.319	1237.06	.593	0.0	-72.5	-12.4	-22.2
Washington D.C.	593	663	13417	2.495	1826.16	.692	14060	.126	1370.28	.625	4.8	-94.9	-24.9	-9.7
London, UK	855	1004	100774	1.834	6062.21	1.943	101985	.157	5263.20	1.916	1.2	-91.4	-13.1	-1.4
Istanbul, TR	693	865	12039	3.071	975.54	.969	12094	.511	685.19	.306	0.4	-83.3	-29.7	-68.3
Perth, AU	532	592	9936	1.637	1146.78	.780	10031	.100	832.50	.341	0.9	-93.8	-27.4	-56.2
Auckland, NZ	1209	1297	21043	2.714	1974.675	.919	20686	.044	1474.30	.347	-1.6	-98.3	-25.3	-62.1
Helsinki, FI	1310	1540	10572	2.287	786.30	.963	10720	.256	669.59	.731	1.4	-88.7	-14.8	-24.0
Vienna, AT	506	586	5334	1.793	799.83	.927	5256	.169	549.12	.350	-1.4	-90.5	-31.3	-62.1
R1	576	1104	1293	2.582	90.292	1.140	1295	.406	67.577	.652	.15	-84.2	-25.1	-42.9
R2	529	1012	1120	2.443	89.538	1.482	1146	.549	61.994	.746	2.3	-77.5	-30.7	-49.6
R3	484	924	1078	2.278	70.247	.824	1077	.474	58.438	.785	-.09	-79.1	-16.8	-4.7
R4	441	840	983	2.451	80.789	1.384	1029	.463	59.075	.882	4.6	-81.1	-26.8	-36.2
R5	400	760	828	2.101	73.852	1.068	856	.385	53.067	.717	3.4	-81.6	-28.1	-32.9
R6	361	684	763	1.734	69.635	1.162	804	.406	50.789	1.013	5.3	-76.5	-27.0	-12.7
R7	324	612	728	2.190	64.192	1.135	780	.538	49.139	.771	7.1	-75.4	-27.0	-12.7
R8	289	544	617	2.063	61.582	1.385	644	.371	42.695	.882	4.3	-82.0	-30.6	-36.2
R9	256	480	534	1.707	57.671	1.106	565	.700	45.705	.800	5.8	-58.9	-20.7	-27.6
R10	225	420	497	1.224	51.627	1.112	523	.329	40.274	.752	5.2	-73.0	-21.9	-32.3
Average											.71	-89.1	-22.4	-38.3

Table 5.5: Solution Quality for Edge Depot Instances with Five Vehicles

			RF				PAR				% Deviation			
Instance	n	m	**Objective**	**ROI**	**ATD**	**HO**	**Objective**	**ROI**	**ATD**	**HO**	**Objective**	**ROI**	**ATD**	**HO**
San Francisco, CA	703	840	9444	1.923	1291.42	1.285	10503	.238	1072.13	.930	11.2	-87.6	-17.0	-27.6
Washington D.C.	593	663	9957	2.551	1836.37	1.000	10519	.221	1152.04	1.320	5.6	-91.3	-37.2	31.9
London, UK	855	1004	65097	1.766	5534.69	3.659	65937	.138	4247.41	3.918	1.3	-92.1	-23.2	7.1
Istanbul, TR	693	865	7461	2.822	760.75	1.200	7825	.555	570.10	.622	2.4	-80.3	-25.0	-48.2
Perth, AU	532	592	6886	2.072	940.74	1.236	6895	.234	754.90	1.152	0.1	-88.6	-19.7	-6.8
Auckland, NZ	1209	1297	14053	2.436	1720.00	1.364	13601	.090	1215.47	.840	-3.2	-96.2	-29.3	-38.4
Helsinki, FI	1310	1540	6806	1.883	729.76	1.327	6833	.288	619.77	.875	0.4	-84.6	-15.0	-34.1
Vienna, AT	506	586	3364	1.187	535.65	.824	3551	.217	430.11	.830	5.5	-81.6	-19.7	0.7
R1	576	1104	833	3.060	81.936	2.232	870	.527	49.017	1.352	4.4	-82.7	-40.1	-39.3
R2	529	1012	735	2.539	70.492	1.978	775	.482	47.565	1.535	5.4	-80.9	-32.5	-22.3
R3	484	924	688	2.668	75.445	2.339	735	.426	44.238	1.668	6.8	-84.0	-41.3	-28.6
R4	441	840	648	2.558	69.623	2.359	694	.523	45.843	1.595	7.0	-79.5	-34.1	-32.3
R5	400	760	567	2.703	61.473	1.892	597	.430	40.702	1.507	5.2	-84.0	-33.7	-20.3
R6	361	684	511	2.329	64.217	2.036	555	.603	39.474	1.502	8.6	-74.0	-38.5	-26.2
R7	324	612	504	2.535	54.606	1.987	545	.650	37.524	1.768	8.1	-74.3	-31.2	-11.0
R8	289	544	417	2.575	53.663	2.081	427	.481	32.781	1.328	2.3	-81.3	-38.9	-36.1
R9	256	480	382	2.399	53.366	2.077	404	.638	32.660	1.297	5.7	-73.4	-38.8	-37.5
R10	225	420	344	1.726	48.749	1.738	370	.493	31.203	1.213	7.5	-71.4	-35.9	-30.1
Average											4.7	-82.6	-30.6	-22.1

Table 5.6: Solution Quality for Edge Depot Instances with 10 Vehicles

			RF				PAR				% Deviation			
Instance	n	m	**Objective**	**ROI**	**ATD**	**HO**	**Objective**	**ROI**	**ATD**	**HO**	**Objective**	**ROI**	**ATD**	**HO**
San Francisco, CA	703	840	6086	2.156	1026.72	2.319	6993	.219	881.76	2.35	14.9	-9.0	-14.1	1.3
Washington D.C.	593	663	9446	3.150	1745.94	2.142	9446	.155	654.25	2.50	0	-95.0	-62.5	16.8
London, UK	855	1004	38986	2.261	4839.18	7.904	42003	.207	3470.18	7.898	7.7	-90.8	-28.2	-0.1
Istanbul, TR	693	865	4483	2.792	655.57	2.056	4577	.540	437.92	1.083	2.1	-80.6	-33.1	-47.3
Perth, AU	532	592	4653	2.306	846.17	2.623	4704	.142	529.20	2.392	1.1	-93.8	-37.4	-8.8
Auckland, NZ	1209	1297	8745	3.248	1612.48	2.277	9211	.138	1102.37	1.845	5.3	-95.7	-31.6	-18.9
Helsinki, FI	1310	1540	4013	2.123	607.46	2.541	4279	.346	514.17	2.333	6.6	-83.6	-15.3	-8.2
Vienna, AT	506	586	2338	1.725	481.73	2.196	2492	.165	311.655	1.65	6.5	-90.3	-35.3	-24.7
R1	576	1104	506	3.641	72.713	3.890	543	.600	35.985	3.070	7.3	-83.5	-50.5	-21.0
R2	529	1012	447	2.895	65.440	3.674	484	.490	33.997	2.965	8.2	-83.0	-48.0	-19.3
R3	484	924	427	3.007	60.911	3.916	474	.611	33.827	4.030	11.0	-79.6	-44.4	2.9
R4	441	840	403	3.100	62.293	3.898	446	.520	32.924	3.234	10.6	-83.2	-47.1	-17.0
R5	400	760	355	3.237	58.904	3.412	391	.537	30.053	3.005	10.1	-83.3	-48.9	-11.9
R6	361	684	320	2.803	55.315	3.692	373	.541	28.309	3.503	16.5	-80.6	-48.8	-5.1
R7	324	612	319	3.652	54.247	4.643	354	.462	27.790	3.195	10.9	-87.3	-48.7	-31.1
R8	289	544	272	2.852	48.756	4.134	312	.508	24.153	2.986	14.7	-82.1	-51.4	-27.7
R9	256	480	251	2.367	47.416	3.286	288	.496	23.317	2.575	14.7	-79.0	-50.8	-21.6
R10	225	420	230	2.232	46.049	3.276	266	.516	22.869	3.339	15.6	-76.8	-50.3	1.9
Average											9.11	-80.9	-41.4	-13.3

by 38.6% on HO. This suggests that PAR would be preferable to use on those instances where planners place a value on the aesthetic quality of their routes.

Over all the instances, the results produced by RF are better than those produced by PAR on the min-max objective. PAR produces better results on the separation and compactness metrics. In Figures 5.13 and 5.14, we show why this is the case. In the solutions produced by RF, adjacent edges are serviced by different vehicles frequently. This causes an increase in ROI since the common endpoint will be counted multiple times. In addition, each route contains edges that are spread out through the graph leading to a suboptimal ATD. Contrast this situation to Figure 5.14 which shows the partition for the same instance used to generate the solution in PAR.

We see that partitions farther from the depot (in the bottom left corner) are noticeably smaller than those close to the depot. Also, there is almost no route overlap between service zones.

In Figures 5.15-5.18, we summarize the relationship between fleet size and the four recorded metrics (i.e., the objective function and the three aesthetic metrics). We see two important trends in the results. First, the difference between the two heuristics appears to grow linearly as a function of the fleet size. We suspect this is the case because, for each trip to and from a partition of the graph, PAR prevents the route from servicing customers who may lie on or close to the path. Since there are $2k$ paths for a fleet size of k (one to the partition, and one returning for each route) we expect that this disadvantage will increase linearly. Second, we see that the line for the centrally located depots is completely beneath the line for the

peripherally located depots. The routes to and from the depot will be longer relative to the sizes of the partitions (or the average load of each vehicle) and, therefore, will lead to a greater difference in the objective function values. It is worth noting that these differences are not seen in the aesthetic metrics because they are already normalized to adjust for the number of routes.

5.7 Conclusions

In this chapter, we developed a cluster-first, route-second heuristic (PAR) for solving the MMKWRPP. PAR produced routes with very appealing visual qualities with only a small increase in the objective function (about 3% on average) as compared to the route-first heuristic (RF).

In the future, it would be interesting to incorporate the aesthetic metrics into the construction or improvement of the routes directly, rather than just measure it in the analysis of the performance of the algorithms. In particular, an approach that includes aesthetic metrics in the calculation of savings to constrain moves made in the improvement and perturbation phases of a traditional algorithm would avoid the computational burden of repeatedly evaluating route-structure metrics (ROI is $O(n)$, ATD is $O(n^2)$). For ROI and ATD, individual moves only have local effects on the value of these two metrics and marginal costs can be used. The next step would be to incorporate these aesthetic measures in a multiobjective function that could be tuned to express the relative preferences of the planner.

Although we have proposed a method for generating routes that conform to the notions of quality in the aesthetic metrics, it is not clear that these metrics actually

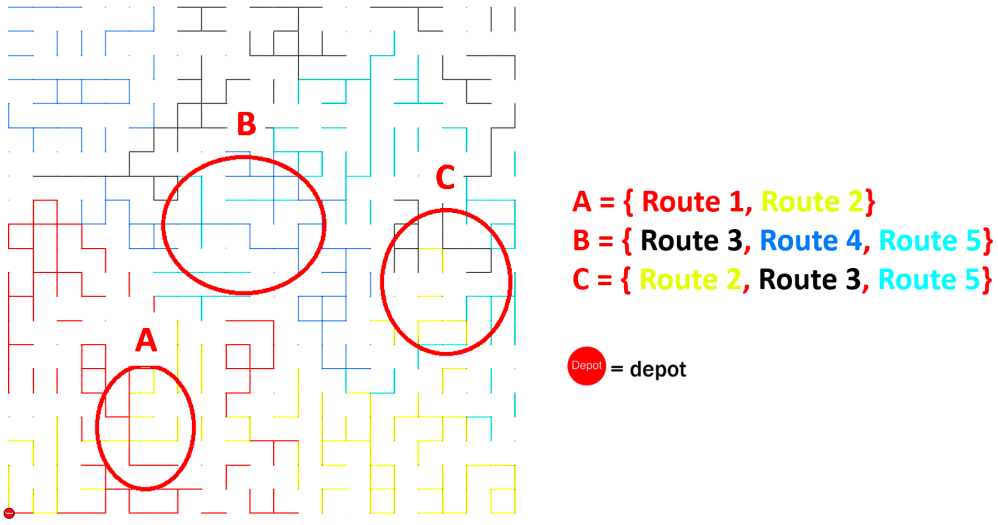


Figure 5.13: One of the solutions generated by RF. Each color corresponds to a single route in the solution. Required edges are colored according to which route services them. Although edges that are not required are not depicted, the underlying network is the filled-in grid. The circled areas violate some of the visual qualities of a good route. For example, multiple trucks are servicing a small, compact region of the network. The list on the right specifies which routes service customers in each of the circled regions.

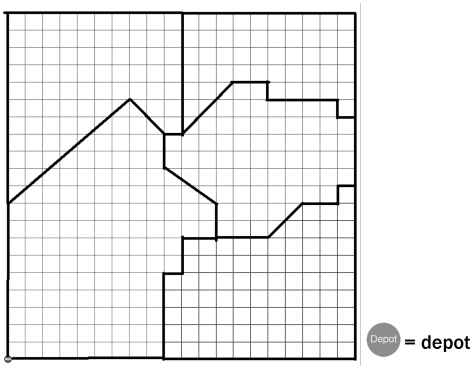


Figure 5.14: The partition for one solution produced by PAR. Each area enclosed by the bold lines corresponds to a single route in the solution. Each vehicle is assigned the customers in one of these regions.

capture the intuitions they were designed to represent. Each metric has a central motivating principle. However, for a given visual quality, there are obviously many ways of attempting to measure it (e.g., ROI and HO for non-overlapping routes).

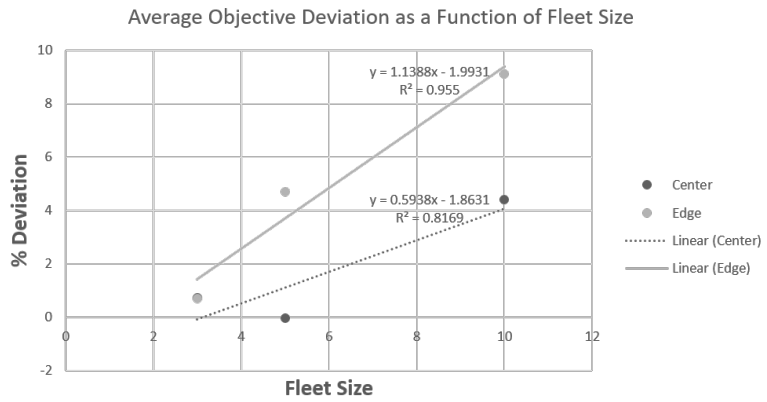


Figure 5.15: Average objective value deviation plotted against the fleet size of the problem instance. Averages are calculated over the 18 instances shown in Tables 5.1 - 5.6. The equation for the trend line is displayed along with its R^2 value.

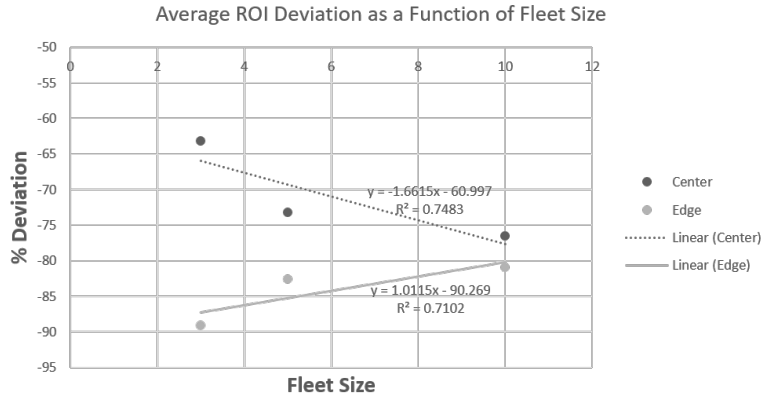


Figure 5.16: Average ROI deviation plotted against the fleet size of the problem instance.

Since the goal is to create routes that would be implemented by a distribution manager, it would be helpful to see how well these metrics match the evaluations of managers. It would be worthwhile to conduct a survey of managers to investigate which metrics correspond to their subjective evaluations of different routing plans.

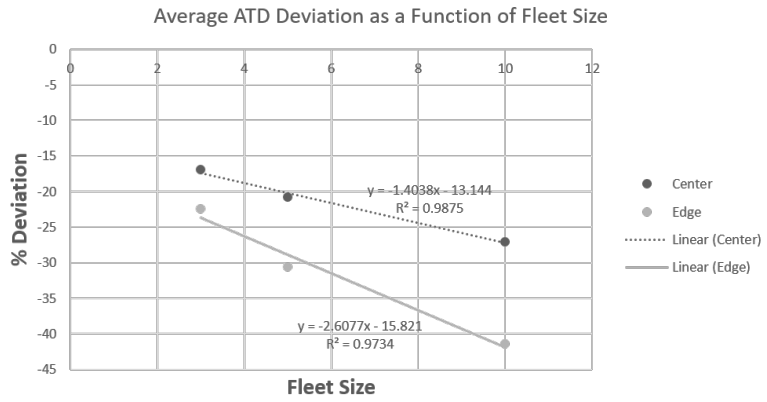


Figure 5.17: Average ATD deviation plotted against the fleet size of the problem instance.

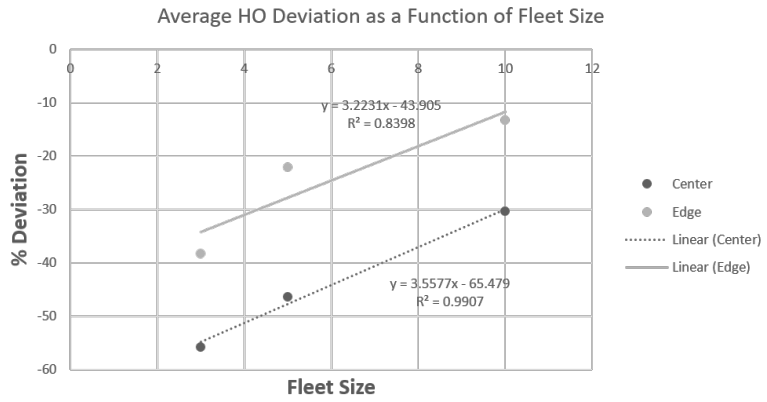


Figure 5.18: Average HO deviation plotted against the fleet size of the problem instance.

Chapter 6: Aesthetic Considerations for the Min-Max K-Windy Rural Postman Problem

6.1 Introduction

Classical routing problems, with objective functions based only on costs or travel distances, produce solutions that are good with respect to the length of the routes but may not be visually appealing. The aesthetic quality of routes is a feature of route planning that is important in the real world since clients sometimes reject optimal solutions provided by optimization algorithms because "they don't look nice". Several practitioners [111] have pointed out that the visual appeal of a proposed set of routes can have a strong influence on the willingness of a client to accept or reject a specific routing plan. Then we could ask: Is it possible to obtain near-optimal solutions that look nicer? And, what is a "nice looking" solution?

For example, in Figure 6.1, we show examples of visually appealing and unappealing routes. The network in (a) has visually appealing routes in which regions of service are compact and separated. Edge color corresponds to a specific route. We see that edges of the same color are clustered together and do not overlap. A set of visually unappealing routes is shown in (b). The routes overlap significantly and each visits nodes and edges that are spread throughout the graph. Panels (c) and (d) are zoomed in views of the areas circled in red in (a) and (b), respectively.

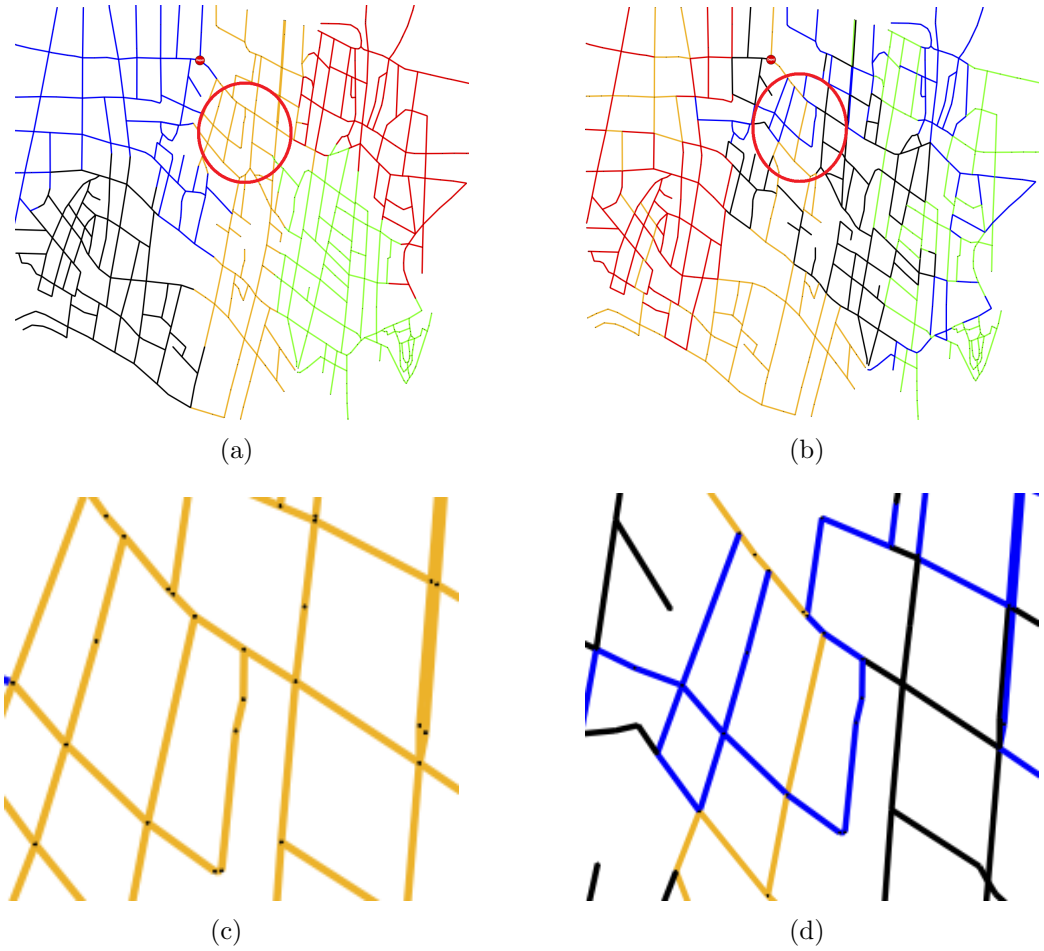


Figure 6.1: Examples of visually appealing and unappealing routes

For Arc Routing Problems, these issues have received relatively little attention in the literature. With respect to visually appealing routes, the most important contribution in the context of this work is due to Constantino et al. [112], where the authors develop several aesthetic metrics, which they refer to as "nice" solution measures. They then study the Mixed Capacitated Arc Routing Problem (MCARP) and two additional variants. The first one is the Non-Overlapping MCARP (NOM-CARP), where the objective function is to minimize the aesthetic metric value. The second is the Bounded Overlapping MCARP (BCARP), where a hard constraint puts a threshold on the maximum aesthetic metric value of feasible solutions, and

the min-max objective function is used. A MIP and heuristic approach are developed for solving the BCARP.

Most relevant to our work in this chapter is the procedure for the the Min-Max K-vehicles Windy Rural Postman Problem (MMKWRPP) in [134]. Given a windy graph with a subset of required edges, the MMKWRPP seeks to find a set of K routes such that each required edge is traversed by at least one vehicle and the cost of the route with maximum cost is minimized. The heuristic in [134] uses an iterative coarsening graph partitioning algorithm from the METIS graph partitioning library [128, 129] to determine the customers assigned to each vehicle. METIS is an open source C library for graph or mesh partitioning released by the University of Minnesota. Since the algorithms contained in the library produce partitions of equal weight, we create a weighting function that will result in balanced routes. The weighting function used as input to the partitioning algorithm takes into account distance from the depot, distance from other customers, and known deadheading requirements. Customers farther from the depot receive higher weights because it is anticipated that servicing them will incur more deadheading. The approach produces initial solutions by using the WRPP heuristic from [122] to route each of the partitions. A set of local search operators is then used to improve the solution by moving customers both between routes and by swapping their position within a route. In order to preserve the visual appeal of the solution, weights and thresholds are used during the improvement phase to prevent significant degradation of the aesthetic quality of the routes.

In this chapter, we consider the MMKWRPP and we study several ways of

incorporating aesthetic measures in the problem. We study this problem because it is a very general arc routing problem. Unlike [134], we consider a model in which both visual quality and cost are included in the objective function. This is the reason why we call the problem the "Multi-Objective Aesthetic MMKWRPP".

Multi-objective optimization is an active field of research, especially for operations research applications like sensor networks [135–137], where the competing objectives are often unrelated, (e.g., coverage, energy consumption of the sensor topology, and network lifetime). Multiple objective functions have also received increased attention in recent years for vehicle routing problems [138, 139], often to explore the tradeoff between cost, fleet size, deviation from a steady plan, route balance, etc. [140, 141] and to handle variants with time windows [141–143].

The remainder of this chapter is organized as follows. In Section 6.1, we present the pure MMKWRPP and an integer programming formulation for it. We review several different aesthetic metrics used in the literature and we focus on two of them particularly suitable for our problem. In Section 6.2, we propose different formulations incorporating these aesthetic measures to the MMKWRPP and we check these models on a set of small-size randomly generated instances. From that discussion, we select a model with a bi-objective function corresponding to what we call the Multi-Objective Aesthetic MMKWRPP. In Section 6.3, we describe a heuristic algorithm that aims to produce visually appealing MMKWRPP solutions of good quality. The computational results presented in Section 6.4 prove that the heuristic obtains quite good solutions on the set of small instances compared with the exact algorithm and we present its performance on a set of large-size instances

taken from benchmark instances for the MMKWRPP. Finally, some conclusions are drawn in Section 6.5.

6.2 The Problem

The Min-Max K-vehicles Windy RPP can be defined as follows. Let $G = (V, E)$ be an undirected and connected graph, where V is the set of vertices and E the set of edges. Let c_{ij}, c_{ji} be the two traversal costs associated with each edge $(i, j) \in E$, one for each direction of traversal, and $E_R \subset E$ a set of edges that must be served (required edges). We denote the depot by $v_1 \in V$, and let K be a fixed number of vehicles. The goal is to find a set of K routes (closed walks starting and ending at the depot) such that each required edge is traversed by at least one vehicle and the cost of the route with maximum cost is minimized.

The MMKWRPP was first studied in [119,144], where a polyhedral study was presented and a branch-and-cut algorithm capable of solving instances with up to 50 vertices, 100 edges, and 4 vehicles was proposed. Later, in [120], a branch-price-and-cut algorithm solving instances with up to 6 vehicles was presented. Both exact algorithms make use of a metaheuristic algorithm that had been described in [121].

The MMKWRPP (see [119]) can be formulated with the following variables:

- for each edge $e = (i, j) \in E$, the variables x_{ij}^k, x_{ji}^k represent the number of times edge e is traversed by vehicle k from i to j or from j to i, respectively,
- for each required edge $e \in E_R$, variable y_e^k takes value 1 if edge e is serviced by the vehicle k and 0 otherwise, and

- a variable z representing the cost of the maximum cost route.

Given a subset $S \subseteq V$, we denote $E(S) = \{(i,j) \in E : i \in S, j \in S\}$, $E_R(S) = E(S) \cap E_R$, $\delta(S) = \{(i,j) \in E : i \in S, j \notin S\}$, $x^k(\delta(S)) = \sum_{(i,j)\in\delta(S)}(x^k_{ij} + x^k_{ji})$, and $x^k(\delta^+(S)) = \sum_{(i,j)\in\delta(S), i\in S} x^k_{ij}$. The Min-Max K-WRPP can be formulated as follows:

$$\text{Minimize} \qquad z$$

s.t.

$$\sum_{(i,j)\in E} \left(c_{ij}x^k_{ij} + c_{ji}x^k_{ji}\right) \leq z \qquad \forall k = 1,\ldots,K \tag{6.1}$$

$$\sum_{k=1}^{K} y^k_e = 1, \qquad \forall e \in E_R \tag{6.2}$$

$$x^k_{ij} + x^k_{ji} \geq y^k_e \qquad \forall e = (i,j) \in E_R \text{ and } \forall k = 1,\ldots,K \tag{6.3}$$

$$\sum_{(i,j)\in\delta(i)} (x^k_{ij} - x^k_{ji}) = 0, \qquad \forall i \in V \text{ and } \forall k = 1,\ldots,K \tag{6.4}$$

$$x^k(\delta(S)) \geq 2y^k_e, \qquad \forall S \subset V \setminus \{1\}, \quad \forall e \in E_R(S), \quad \forall k = 1,\ldots,K \tag{6.5}$$

$$x^k_{ij}, x^k_{ji} \geq 0 \text{ and integer} \qquad \forall (i,j) \in E, \quad \forall k = 1,\ldots,K \tag{6.6}$$

$$y^k_e \in \{0,1\} \qquad \forall e \in E_R \quad \forall k = 1,\ldots,K. \tag{6.7}$$

Inequalities (6.1) imply that the maximum cost vehicle route is minimized. Equations (6.2) ensure that each required edge is serviced by exactly one vehicle and traversing inequalities (6.3) force a vehicle to traverse the edges it services. Symmetry equations (6.4) force each vehicle route to be symmetric, while connectivity inequalities (6.5) ensure that each vehicle route connects the edges it services and the depot.

As noted in the introduction, solutions for the above model can be unsatisfactory in terms of aesthetic criteria. With the goal of improving the solutions from an aesthetic point of view, we will incorporate some constraints to the previous formulation or modify its objective function.

Several metrics have been proposed for measuring the visual quality of a route. A summary of these measures is given in Table 6.1. Most of these metrics emphasize several key attributes of aesthetically appealing routes. They encourage non-overlapping, compact, contiguous routes. Together, these properties tend to produce plans that are both efficient (in terms of distance) and driver-friendly, and allow last-minute adjustments (cancellations or urgent service calls which require an online response) to be made without significantly disturbing the routes or the drivers.

Of all the metrics shown in Table 6.1, in this chapter we consider two of them. The first one is the Routes Overlapping Index (ROI), proposed by Constantino et al. [112], which counts the number of nodes that are visited by more than one route:

$$\text{ROI} = \frac{NO - |V|}{(\sqrt{K} + \sqrt{|V|} - 1)^2 - |V|}, \tag{6.8}$$

where node overlap (NO) is $\text{NO} = \sum_{i \in V} \sum_{k=1}^{K} n_i^k$, where n_i^k takes value 1 if vertex i belongs to route k. . The second measure we will study, also proposed in [112], is the Average Task Distance (ATD):

$$\text{ATD} = \frac{1}{K} \sum_{k=1}^{K} \frac{\sum_{e_1, e_2 \in E_R \text{ served by } k} d_{e_1 e_2}}{|taskpairs|}, \tag{6.9}$$

where $d_{e_1 e_2}$ is the minimum length of the shortest paths between the end nodes of

Notation		
Metric	Description	Reference
N_h	Number of vertices in the convex hull of multiple routes	[115, 131]
CLP	The crossing length percentage. Crossing points within the context of a single route are identified. Each crossing point divides the route into two cycles. The minimum length of the two is defined as the crossing length of the crossing point. The CLP is the sum of the crossing lengths divided by the length of the route.	[113]
A_o	Average number of outliers on a route, where outlier is defined as vertices closer to the center of a route they are not on.	[115]
N_m	Number of vertices closer to another route's median (the vertex that minimizes the maximum distance to any other vertex on the tour) than its own.	[117]
COMP	$\frac{\text{AvgDistance}}{\text{AvgMaxDistance}}$ where AvgDistance and AvgMaxDistance are the average distance between consecutive customers, and the average distance restricted to the greatest 20% of consecutive distances, respectively.	[114]
DGRB	$2 * \left(1 - \frac{\lvert\hat{O}\rvert}{\lvert O\rvert}\right) - 1$, where $\lvert\hat{O}\rvert$ and $\lvert O\rvert$ are the number of customers closer to another route's center (of gravity) and the number of customers on the route, respectively. We note that $\lvert\hat{O}\rvert = \lvert A_o\rvert$.	[114]
VA	$\frac{1}{\frac{\text{NC}}{5} + \frac{1}{\text{COMP}} + \frac{1}{\text{DGRB}} - 1}$, where NC is the number of crossings within a route.	[114]
ROI	The route overlap index. $\frac{\text{NO} - \lvert V\rvert}{(\sqrt{K} + \sqrt{\lvert V\rvert} - 1)^2 - \lvert V\rvert}$, where K is the number of routes. $\lvert V\rvert$ is the number of vertices in the graph, including the depot. $\text{NO} = \sum_{i \in V} \sum_{k=1}^{K} n_i^k$, where n_i^k takes value 1 if vertex i belongs to route k.	[112]
ATD	The average task distance. $\frac{\sum_{k=1}^{K} \sum_{a,b \text{ served by } k} \text{Dist}_{ab}}{\lvert\text{taskpairs}\rvert}$, where $\lvert\text{taskpairs}\rvert = \frac{\lvert E_R\rvert * (\lvert E_R\rvert - K)}{2K}$ is the approximate average number of tasks assigned to each route.	[112]
CI	The connectivity index. $\frac{CC}{K}$, where CC is the number of connected components of the required edges in the candidate solution.	[112]
HO	The hull overlap metric. $\sum_{1 \le k_1 \ne k_2 \le K} \frac{\text{intersec}(\text{convex}(k_1), \text{convex}(k_2))}{\text{convex}(k_1)} \frac{1}{K}$, where convex$(k)$ is the convex hull of the vertices in route k and intersec(H_1, H_2) is the area of the intersection between hulls H_1 and H_2.	[134]

Table 6.1: Metrics for assessing the visual quality of routes.

required edges e_1 and e_2, and $|taskpairs| = \frac{|E_R| * (|E_R| - K)}{2K}$ is an estimate of the average number of pairs of required edges in a single route.

The measures in Table 6.1 fall into several categories. The first category uses convex hulls to measure route overlap. A second uses proximity to route centers (i.e., medians or geographic centers) to measure compactness. However, there are clear instances where aesthetically good routes can perform poorly according to these measures (e.g., routes that are nested circles). ROI does not have this problem. A

third category uses route crossings (places where a route intersects with itself) to assess visual quality. These measures are better-suited to node routing applications where travel between nodes is abstracted as straight lines between nodes. Since we are interested in solving problems over street networks, these crossings can depend on how the (potentially same) network is represented. For example, simplifications which do not change distances or connectivity can change the number or location of crossings. The last category uses deadheading time to measure compactness. We select one of these to study (ATD).

In the following section, we study several ways of incorporating these ROI and ATD metrics to the previous formulation of the MMKWRPP.

6.3 Incorporating Aesthetic Metrics

A metric similar to the ROI can be incorporated into the formulation as follows. Let V_R be the set of vertices in $V \setminus \{v_1\}$ incident with a required edge, and let $V_{NR} = V \setminus \{V_R \cup \{v_1\}\}$. For each $i \in V \setminus \{v_1\}, k = 1, \ldots, K$, we define the variables

$$w_i^k = \begin{cases} 1 & \text{if vertex } i \text{ is visited by route } k, \\ 0 & \text{otherwise,} \end{cases}$$

and add the following constraints to the formulation:

$$w_i^k \leq x^k(\delta^+(i)) \leq M w_i^k, \quad \forall i \in V \setminus \{v_1\}, k = 1, \ldots, K.$$

We also define, $\forall i \in V_{NR}$, the variable $\bar{w}_i$ as the number of routes that visit vertex i minus 1 if it is visited by more than one route, $\bar{w}_i = 0$ otherwise, and add

the constraints:

$$\bar{w}_i \geq \sum_{k=1}^{K} w_i^k - 1 \quad \forall i \in V_{NR}.$$

Then we define the measure RO (Routes Overlap) as:

$$\text{RO} = \sum_{k=1}^{K} \sum_{i \in V_R} w_i^k + \sum_{i \in V_{NR}} \bar{w}_i - |V_R|.$$

Note that $\text{RO} = NO - |V|$ is proportional to the ROI defined in (6.8). We have incorporated the RO measure to the MMKWRPP formulation in three different ways, producing three different models that will be called M1, M2, and M3.

Model M1 incorporates the following constraint:

$$\text{RO} \leq \text{RO}_{crit},$$

where RO_{crit} is the maximum number of overlapping vertices that we allow the solution to contain. In order to decide the value of RO_{crit} for a given instance, we solve the MMKWRPP, calculate the number of overlapping vertices in the optimal solution, and divide this number by 2. If model M1 is infeasible for this value of RO_{crit}, we increase it iteratively until a feasible solution is obtained.

In Model M2 we change the objective function to

$$\min \quad \text{RO}$$

and add the following constraint to the formulation

$$z \leq (1 + p_z) z_{opt},$$

where z_{opt} is the length of the longest route in the optimal MMKWRPP solution and $0 \leq p_z \leq 1$ is a parameter representing the maximum increase allowed for z.

Model M3 uses a multi-objective function:

$$\min \quad \alpha z + (1-\alpha)\ \mathrm{RO}\ \frac{z_{opt}}{\mathrm{RO}_{crit}}$$

where z_{opt} and RO_{crit} are calculated as in models M2 and M1, respectively, and $\alpha \in [0,1]$ is a parameter. Note that if the values of z and RO are similar to z_{opt} and RO_{crit}, respectively, then both terms in the objective function are of comparable magnitude and therefore α represents the relative weight of each objective.

After some computational tests, we observed that:

- The aesthetic quality of the solutions of Model M1 depends strongly on the value of RO_{crit} and obtaining an appropriate value is not trivial.
- While z_{opt} for Model M2 is easy to obtain for small and medium size instances (we can solve the MMKWRPP efficiently), this model was harder to solve than the other models.

Therefore we chose to work with model M3.

In order to evaluate the effect of α, we have used model M3 to solve an instance based on part of the street network of Paris, with 2, 3, and 4 vehicles. This Paris instance has 83 vertices and 86 edges, 44 of which are required. Table 6.2 shows the values of z and RO obtained with different values of α with the branch-and-cut algorithm described in [144] and adapted to model M3. All of the values, except the ones marked with an asterisk, correspond to optimal solutions.

The case $\alpha = 1$ corresponds to the pure MMKWRPP objective and does not take into account the value of RO. Whereas the opposite case, $\alpha = 0$, produces

	K=2		K=3		K=4	
α	Z	RO	Z	RO	Z	RO
1	2245	3	1651	15	1403*	23*
0.99	2245	2	1651	13	1403*	16*
0.9	2245	2	1767	5	1413	15
0.75	2472	1	1767	5	1413	15
0.5	2472	1	2245	2	1767	6
0.25	3926	0	2245	2	2245	3
0.1	3926	0	3926	1	2245	3
0.01	3926	0	3926	1	3926	2
0	4595	0	4121	1	5097	2

Table 6.2: Effect of parameter α on Paris instance

solutions with very low values of RO but at the expense of very long routes. Note that optimizing only the RO value (i.e., $\alpha = 0$) is not very meaningful from a practical perspective. This is because any solution that only uses a single vehicle will completely avoid overlap. Furthermore, there is no incentive to minimize the length of this route because it has no impact on RO. While we ensure that every vehicle is used, this does not help avoid the issue. For example, if there are two vehicles, a solution where one vehicle services a single customer and the other services every other customer is likely to minimize RO, regardless of how many other customers there are. Figure 6.2 provides a concrete example of this. The route of each vehicle is depicted with a different color. Solid lines represent arcs that are served, while dotted lines are arcs traversed in deadheading. It shows an optimal solution to the Paris instance with a fleet size of 2 when $\alpha = 0$. The routes are very unbalanced (one costs 570, another costs 4595). In addition, the longer route is not the least cost way of servicing the customers assigned to the vehicle. However, since there is

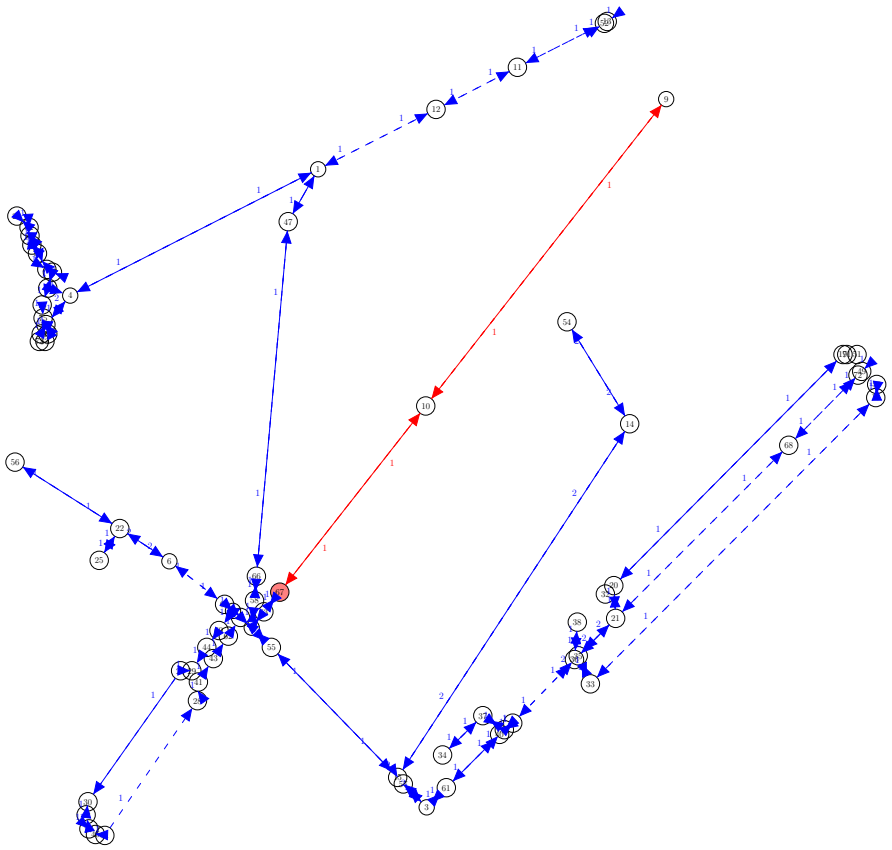


Figure 6.2: Optimal solution of Paris instance with $K = 2$ and $\alpha = 0$

no contribution of cost to the objective function, this inefficiency does not affect the quality of the solution.

If we consider, for example, the case with three vehicles, we can see that the length of the longest route varies from 1651, when $\alpha = 1$, to 4121, when $\alpha = 0$ and we only minimize the number of overlapping vertices. Note that the solutions in these cases are dominated by other solutions obtained with different values of α. In between these extreme values, we can find solutions with a better balance

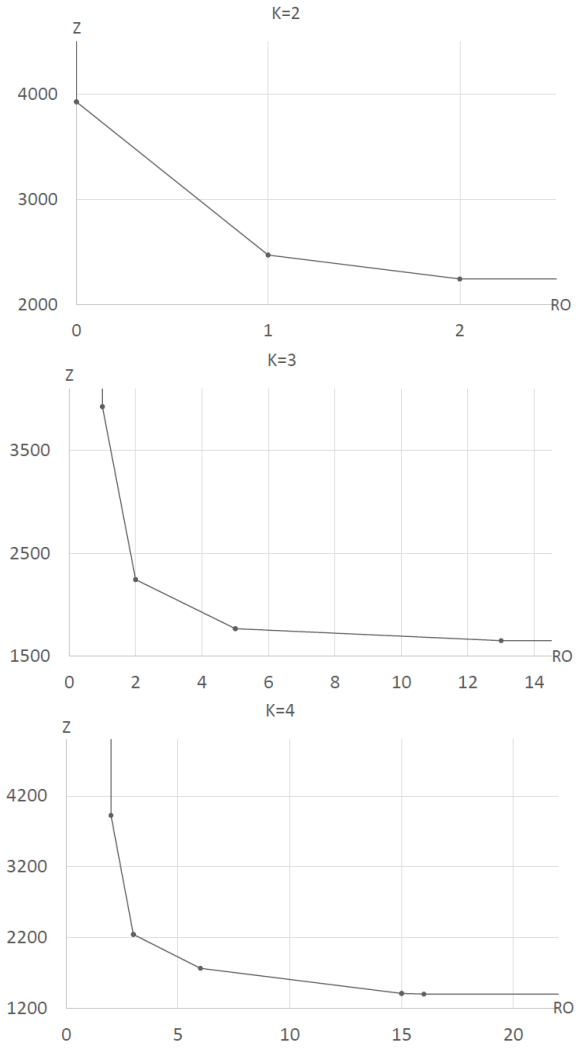
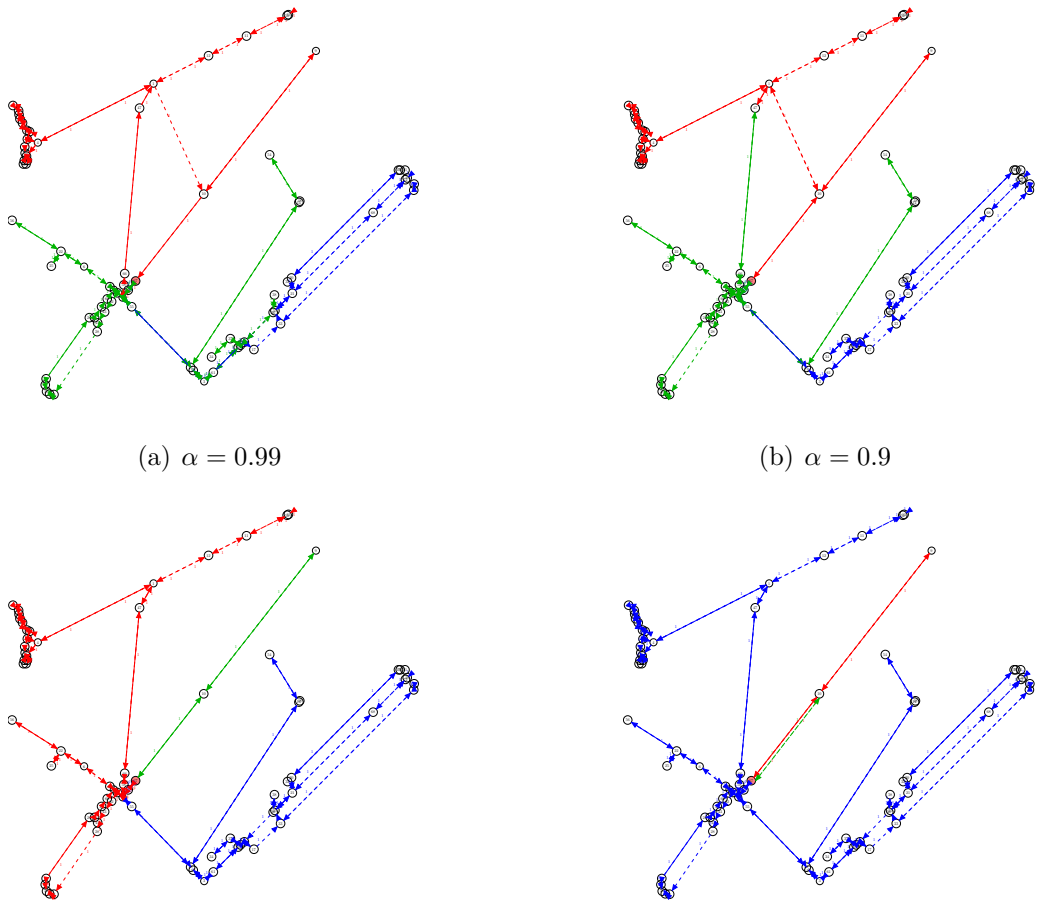


Figure 6.3: Pareto front of Paris instance with RO

between the maximum length and the routes overlapping that could be considered more appealing from a practical point of view.

The effect of α in the optimal solution can also be appreciated on the Pareto fronts depicted in Figure 6.3. Each point on the curve represents a solution for which it is impossible to improve either objective (z or RO) without decreasing performance with respect to the other.

Figure 6.4 shows the optimal solutions of the Paris instance with $K = 3$



(a) $\alpha = 0.99$ (b) $\alpha = 0.9$

(c) $\alpha = 0.5$ (d) $\alpha = 0.01$

Figure 6.4: Optimal solutions of Paris instance with $K = 3$ and different values of α

vehicles and four different values of α. As before, the route of each vehicle is depicted with a different color. Solid lines represent arcs that are served, while dotted lines are arcs traversed in deadheading. The solution corresponding to $\alpha = 0.99$ has RO=13 and the lengths of the three routes are 1651, 1638, and 1527. When $\alpha = 0.9$, RO=5 and the lengths are 1767, 1604, and 1585. For $\alpha = 0.5$ we have RO=2 and lengths 2245, 1723, and 570. Finally, for $\alpha = 0.01$, RO=1 and the lengths are 3926, 570, and 229. As expected, the lower the value of RO, the greater the length of the longest route (which causes the routes to be more unbalanced).

Regarding the ATD metric (6.9), it can be calculated by using y variables as

$$ATD = \frac{1}{|K|}\sum_{k=1}^{K}\frac{\sum_{e_1,e_2\in E_R} y^k_{e_1}y^k_{e_2}d_{e_1e_2}}{|taskpairs|}.$$

However, this expression is not linear. In order to linearize it, we need to add more variables and constraints to the formulation. For each pair of required edges $e_1, e_2 \in E_R$ and each vehicle $k = 1, \ldots, K$, we add a binary variable $t^k_{e_1e_2}$ and the following constraints:

$$t^k_{e_1e_2} \leq y^k_{e_1}, \quad t^k_{e_1e_2} \leq y^k_{e_2}\ , \ \text{and} \quad t^k_{e_1e_2} \geq y^k_{e_1} + y^k_{e_2} - 1.$$

Then, the ATD can be calculated as

$$ATD = \frac{1}{|K|}\sum_{k=1}^{K}\frac{\sum_{e_1,e_2\in E_R} t^k_{e_1e_2}d_{e_1e_2}}{|taskpairs|}.$$

Note that this implies adding $|K||E_R|^2$ variables and $3|K||E_R|^2$ constraints to the ILP. Therefore, the problem becomes harder to solve than the one using the RO metric. Hence, we have studied the use of the ATD metric only in a multi-objective model similar to model M3, with the objective function

$$\min \quad \alpha z + (1-\alpha)ATD\frac{z_{opt}}{ATD_{est}},$$

where ATD_{est} is an estimate of ATD obtained from the solution of the MMKWRPP.

As with RO, in order to evaluate the effect of α in the multi-objective model with the ATD metric, we have solved an instance based on part of the street network of San Francisco, with 2, 3, and 4 vehicles. This San Francisco instance has 52

	K=2		K=3		K=4	
α	Z	ATD	Z	ATD	Z	ATD
1	1824	103.74	1243	93.78	1031*	92.14*
0.99	1824	103.74	1243	90.70	1031	89.67
0.9	1824	103.74	1243	90.70	1031	89.67
0.75	1876	91.08	1243	90.70	1031	89.67
0.5	1876	91.08	1243	90.70	1135	71.14
0.25	1876	91.08	1326	85.62	1155	69.22
0.1	1876	91.08	2073	77.75	1248	67.17
0.01	1876	91.08	2073	77.75	1406	66.10
0	4151	91.08	5722	77.75	7824	66.10

Table 6.3: Effect of parameter α on San Francisco instance

vertices and 55 edges, 29 of which are required. The values of z and ATD obtained are shown in Table 6.3. All of the values, except the one marked with an asterisk, correspond to optimal solutions. Figure 6.5 shows the associated Pareto fronts. As with the RO metric, the solutions corresponding to the extreme values of α are dominated.

Figure 6.6 illustrates the effect of α in the San Francisco instance. The solution associated with $\alpha = 0.99$ has ATD=89.67 and route lengths 1031, 1016, 1013, and 991, while the one corresponding to $\alpha = 0.01$ has ATD=66.10 and lengths 1406, 1178, 825, and 716. Note that the routes obtained with $\alpha = 0.01$ seem to be more compact since they serve edges that are closer to each other.

It is interesting to note that the RO value of the solution with $\alpha = 0.01$ is 10, while the RO of the other one, which has a bigger ATD, is 9. Therefore, we decided to compare the visual effect of these two metrics. After observing the solutions obtained for several instances, we noticed that the difference in the effect of RO and

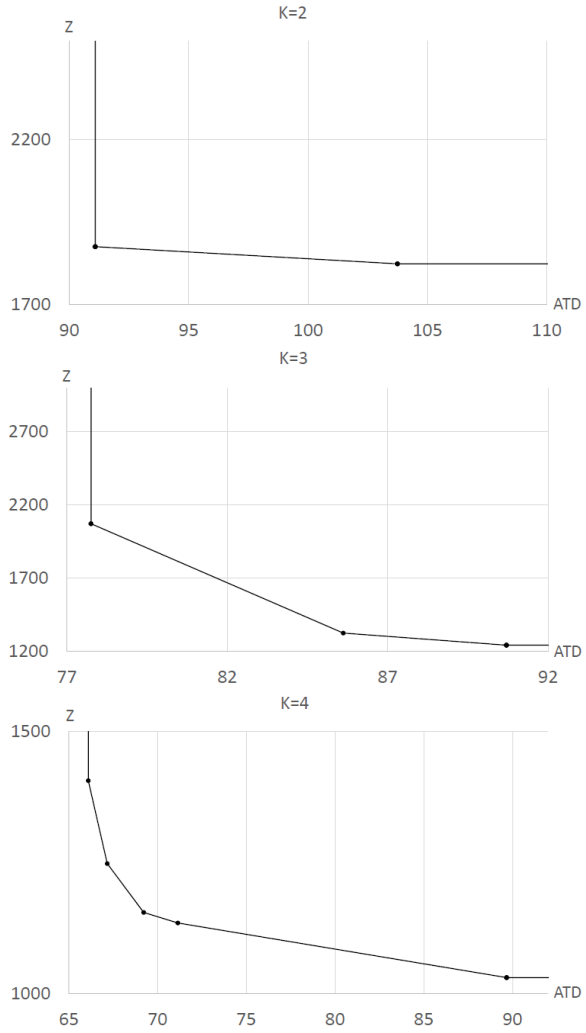
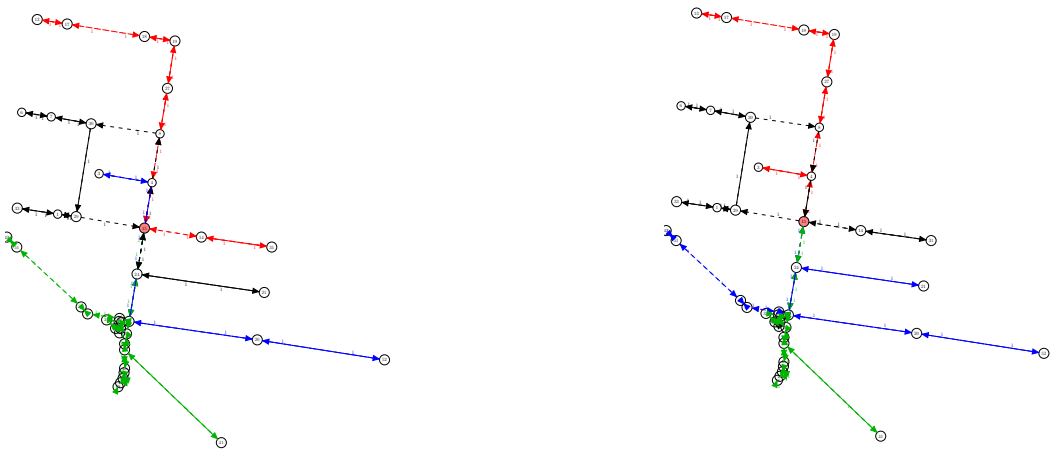


Figure 6.5: Pareto front of San Francisco instance with ATD

ATD was influenced by the topology of the instance graphs. Therefore, we decided to study this effect on instances based on a grid graph and generated as follows: V is defined by all the vertices in a $r \times s$ grid and the depot is chosen among them randomly. E contains all the edges joining adjacent vertices in the grid (vertically and horizontally), and some "diagonal" edges randomly chosen. The cost of each edge is calculated as the Euclidean distance between its endnodes multiplied by a random number in $[0.85, 1.15]$ rounded to the nearest integer. Each edge is selected



(a) $\alpha = 0.99$ (b) $\alpha = 0.01$

Figure 6.6: Optimal solutions of San Francisco instance with $K = 4$ and different values of α

as required with probability $p \in \{0.3, 0.5, 0.7\}$. Table 6.4 gives the characteristics of the generated instances, where, for example, the instance Grid_7_4_3 has been generated on a 7×4 grid and $p = 0.3$.

After comparing the solutions obtained using both RO and ATD, we observed that the two metrics produced solutions that are different but it was not possible to conclude that one metric is better than the other with respect to aesthetic considerations. See Figure 6.7, for example. In this figure, the routes of the solution obtained considering the RO metric do not overlap, while the arcs serviced by the routes in the other solution define a more compact region (although the full routes, including deadheading, are not very compact). The solution on the left has RO=7, ATD=134.9, and route lengths 1980 (blue), 1484 (red), and 2008 (green), while the one on the right has RO=15, ATD=91.4, and route lengths 2005 (blue), 1963 (red),

Instance	$\|V\|$	$\|E\|$	$\|E_R\|$
Grid_5_5_3	25	50	24
Grid_5_5_5	25	50	30
Grid_5_5_7	25	50	38
Grid_6_5_3	30	63	25
Grid_6_5_5	30	63	34
Grid_6_5_7	30	63	45
Grid_7_4_3	28	57	24
Grid_7_4_5	28	57	32
Grid_7_4_7	28	57	41
Grid_7_6_3	42	94	34
Grid_7_6_5	42	94	48
Grid_7_6_7	42	94	65

Table 6.4: Characteristics of the Grid instances

and 1987 (green). It should be clear that both solutions are aesthetically appealing.

6.4 Heuristic

We now review the heuristic from Lum et al. [134] and describe the modifications used to improve its performance on the multi-objective problem. This heuristic assigns a score to each customer according to the following formula:

$$b_{ij} = \begin{cases} 1 & \text{if } (i,j) \text{ is in } E_{multi} \subset E, \text{ the set of edges identified by the deadhead subroutine.} \\ 0 & \text{otherwise} \end{cases} \tag{6.10}$$

$$c_{ij}^{new} = \begin{cases} (1 + b_{ij}) * c_{ij} + \alpha * \dfrac{2 * \min\{\text{Dist}(0,i), \text{Dist}(0,j)\}}{\dfrac{|E_R|}{K}} + \text{minReqDist}(i,j) & \forall (i,j) \in E_R \\ 0 & \text{otherwise} \end{cases} \tag{6.11}$$

where α is a parameter that affects the distance penalty relative to the original edge

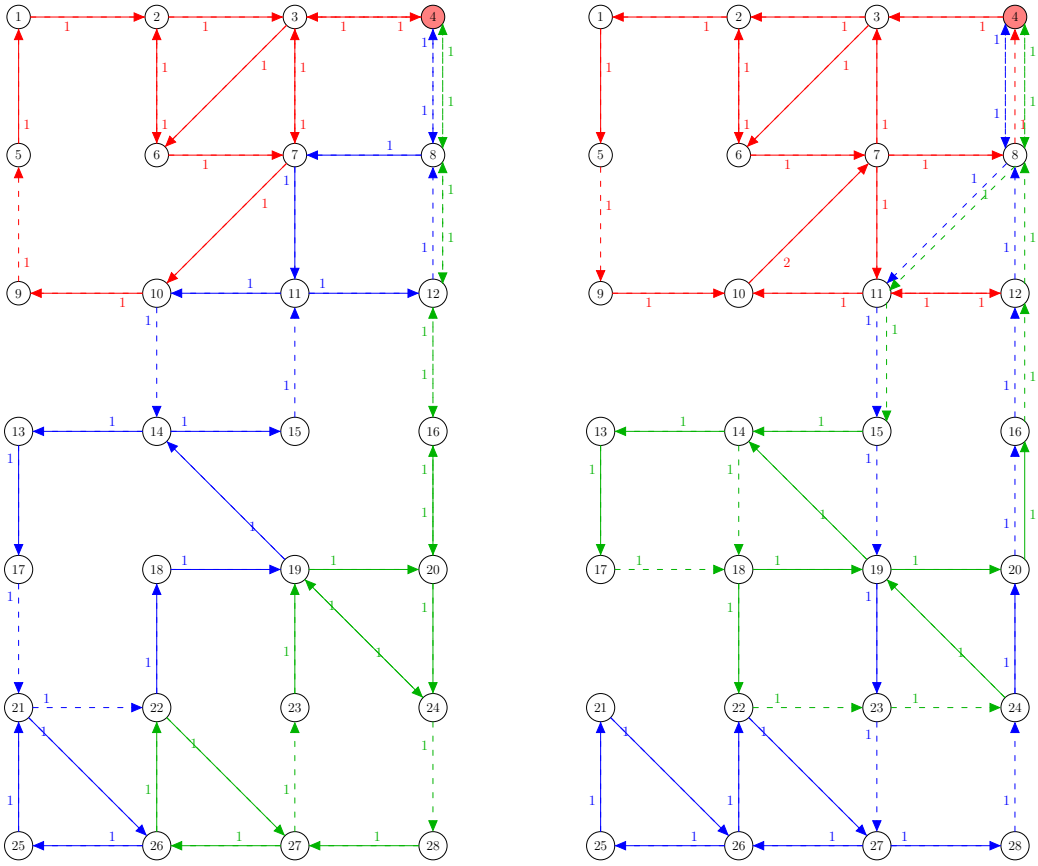


Figure 6.7: RO (left) vs ATD (right) solutions for the instance Grid_7_4_5 with $\alpha = 0.75$

cost, and minReqDist$((i, j))$ is a function that generates the shortest distance from (i, j) to any other edge in E_R.

The first term in c_{ij}^{new} penalizes edges in the graph that will have to be traversed multiple times (deadheaded) in the final solution, where c_{ij} is the original cost of (i, j) in G. The deadhead algorithm is a preprocessor that identifies edges in the final solution that will have to be traversed multiple times. The second term in c_{ij}^{new} penalizes each partition by the distance to the depot. The intuition is that $|E_R|/K$ estimates the number of edges in each partition. The factor of two incorporates the path to and from the partition. If the number of required edges in each partition is

nearly equal, a value of α near 1 should produce good results. Finally, the last term in c_{ij}^{new} penalizes each required edge by the minimum distance required to reach another required edge (or the depot). After servicing a street, a vehicle will need to travel at least this distance to continue the route.

The METIS graph partitioning open source software is then used to assign customers to vehicle routes. A heuristic from Benavent et al. for the single vehicle WRPP [122] is then used to determine the route of each vehicle. The routes are then improved using three single route improvement procedures, as well as three multi-route improvement procedures organized according to a procedure outlined in [121]. The improvement procedures swap strings of customers and determine the optimal direction of traversal for each of the customers.

We now explain the changes made to the algorithm to improve its performance for the multi-objective problem. The first modification is to the multi-route improvement procedures. Previously, moves were made based on a cost savings calculation that only included the effect on the min-max objective function and, therefore, it was appropriate to only consider moving customers from the longest route onto others. (Every other move would have a savings of zero). In the multi-objective setting, we alter the procedures to consider the weighted multi-objective function and now consider moves between all pairs of routes.

The second modification we make adds a perturbation procedure to the improvement phase of the heuristic. Once we have completed an iteration of local search, the center of each route is calculated according to the formula $(\sum_{v_i \in R_k} \frac{x_i}{|R_k|}$,

$\sum_{v_i \in R_k} \frac{y_i}{|R_k|}$), where R_k is the set of vertices visited by vehicle k in the current working solution. This is simply the center of mass if each node on the route is a point mass. A new set of route centers is calculated by rotating each point by a fixed angle. Each customer is then assigned to the new center point closest to it and the local search operators are executed to improve this solution. This loop is repeated for a fixed number of iterations (10, in our case). The angle of rotation is chosen so that each center makes a full rotation in the course of the improvement phase (i.e., 36 degrees). This is depicted in Figure 6.8. The route rotation perturbation procedure operates directly on the geographic centers of the routes. The figure is an example of a two-vehicle problem. The solid roads connect customers assigned to one route, and the dotted roads connect customers assigned to the second route. Before each rotation, the centers of each route are computed (depicted by the stars). These are rotated and customers are reassigned to the closest center.

This second modification is inspired by the work of Tang and Miller-Hooks, who introduce the median of a tour (the vertex that minimizes the maximum distance to any other vertex on the tour) and give several metrics for compactness and overlap [117]. They propose the total number of vertices closer to another route's median (similar to Poot's measure [115], but substituting the location of the median for the geographic center). The authors show that if every node is closer to its route's median than any of the other medians, then their convex hulls will not overlap. An example is shown in Figure 6.9. In (a), there are three routes. The red vertices are the medians of each route. Every vertex is closer to its own route

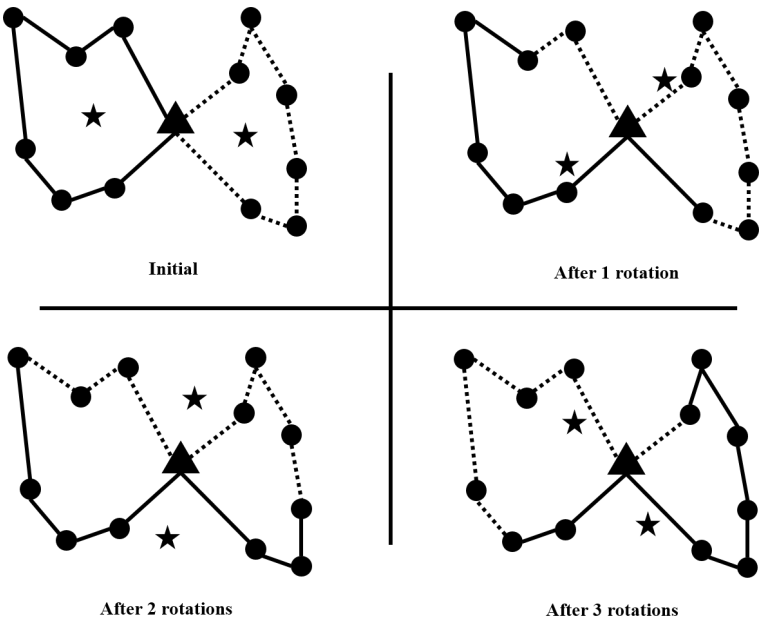


Figure 6.8: The route rotation perturbation procedure.

median than any of the other medians. The convex hulls of the routes do not overlap at all. In (b), the two white vertices are closer to the median of another route. The convex hulls of the routes are overlapping. We take this as a proxy for a set of routes having good aesthetic qualities.

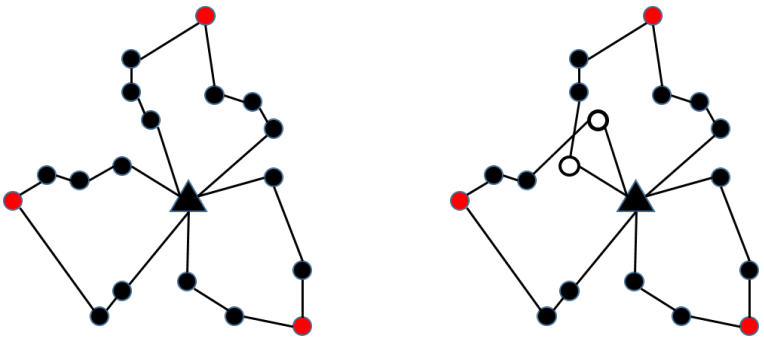


Figure 6.9: An example illustrating the result of Tang and Miller-Hooks [117].

6.5 Computational Results

In this section we present some results for both the exact and heuristic methods. We first compare results using RO versus ATD in the objective function of the exact approach. We show that using RO allows us to solve more instances and get closer to optimality, when we cannot solve the problem exactly. Then, we compare the heuristic approach to the exact approach on a series of smaller test instances. Finally, we show the scalability of the heuristic approach by solving a series of large test instances.

Table 6.5 shows the results obtained with the branch-and-cut algorithms for Model M3 with RO and with ATD on the set of randomly generated instances described in Table 6.4. The results associated with instances of the same size have been grouped together and shown in each block of the table. Each row corresponds to a different value of $\alpha \in \{0.01, 0.1, 0.25, 0.5, 0.75, 0.9, 0.99\}$ and gives the number of optimal solutions found out of three and the average gap of the unsolved instances for the two models and for two, three, and four vehicles. The exact algorithm was coded in C++ and ran on a Intel i7-4770 machine with 32GB RAM using Cplex 12.6. The time limit was set to two hours.

We can observe that almost all the instances with two vehicles have been solved to optimality with both models, as well as many of the instances with three vehicles associated with the small and medium size instances. This is not the case with four vehicles, where the branch-and-cut algorithms have not been able to solve medium and large size instances optimally. The tendency that branch-and-cut algorithms

		K=2				K=3				K=4			
		RO		ATD		RO		ATD		RO		ATD	
Inst.	α	Opt	Gap	Opt	Gap	Opt	Gap	Opt	Gap	Opt	Gap	Opt	Gap
	0.01	3	-	3	-	3	-	2	9.3	3	-	2	35.7
	0.1	3	-	3	-	3	-	2	11.9	3	-	2	43.2
	0.25	3	-	3	-	3	-	2	17.6	1	2.0	2	33.3
5_5	0.5	3	-	3	-	2	1.5	2	12.9	1	12.3	2	29.4
	0.75	3	-	3	-	2	2.6	2	6.3	0	9.1	1	9.2
	0.9	3	-	3	-	3	-	2	3.4	1	5.6	2	7.0
	0.99	3	-	3	-	3	-	2	0.3	2	0.3	2	0.4
	0.01	3	-	3	-	3	-	2	27.7	2	3.8	2	53.9
	0.1	3	-	3	-	3	-	2	22.5	2	5.0	2	42.5
	0.25	3	-	3	-	3	-	2	24.3	1	5.7	1	26.1
7_4	0.5	3	-	3	-	2	3.7	2	15.4	0	9.8	1	16.9
	0.75	3	-	3	-	1	5.5	2	5.4	0	7.6	1	11.0
	0.9	3	-	3	-	1	3.9	3	-	0	6.8	1	6.6
	0.99	3	-	3	-	3	-	3	-	1	5.3	2	6.4
	0.01	3	-	3	-	2	3.6	1	22.9	0	3.6	1	55.3
	0.1	3	-	3	-	2	2.8	1	19.8	0	7.6	1	44.9
	0.25	3	-	3	-	2	10.2	1	20.0	0	11.6	1	42.0
6_5	0.5	3	-	3	-	2	9.0	1	14.0	0	14.5	1	26.8
	0.75	3	-	3	-	1	5.1	2	6.3	0	12.8	1	15.2
	0.9	3	-	3	-	1	3.6	2	2.8	0	6.4	2	7.6
	0.99	3	-	3	-	3	-	3	-	2	3.2	2	2.0
	0.01	1	2.4	3	-	0	3.4	0	51.7	0	21.3	0	72.4
	0.1	3	-	3	-	0	4.2	0	50.2	0	12.7	0	62.5
	0.25	3	-	2	5.6	0	7.2	0	42.4	0	14.5	0	57.0
7_6	0.5	3	-	2	4.4	0	8.7	0	28.4	0	24.5	0	36.5
	0.75	3	-	2	2.9	0	10.6	0	16.8	0	20.2	0	21.9
	0.9	3	-	2	2.2	1	11.0	1	8.5	0	18.9	0	11.4
	0.99	3	-	2	0.3	1	2.2	1	1.2	0	10.6	1	6.3

Table 6.5: Comparison of multi-objective models with RO and ATD for Grid instances

		RO				ATD			
	alfa	Z	RO	ATD	Time	Z	RO	ATD	Time
	0.01	2607.7	0.3	122.9	2.2	2485.7	4.0	113.0	6.9
	0.1	2607.7	0.3	122.9	2.3	2361.7	3.3	113.4	8.6
	0.25	2607.7	0.3	122.9	2.8	2288.3	3.0	114.2	15.7
5_5	0.5	2607.7	0.3	122.9	4.1	2288.3	3.3	114.2	16.5
	0.75	2360.0	1.0	116.8	6.2	2288.3	3.0	114.2	21.6
	0.9	2298.3	1.3	114.3	8.9	2194.7	5.3	150.5	33.5
	0.99	2185.0	6.0	179.2	8.6	2180.7	8.7	178.7	37.2
	0.01	3121.7	0.3	180.8	9.9	2597.7	3.7	118.9	5.5
	0.1	3121.7	0.3	180.8	4.4	2597.7	4.3	118.9	7.4
	0.25	3121.7	0.3	180.8	3.3	2514.0	4.0	119.8	10.5
7_4	0.5	2768.7	1.0	153.8	12.0	2459.3	4.0	121.3	19.8
	0.75	2429.0	1.7	125.8	12.2	2410.0	5.7	129.4	41.5
	0.9	2429.0	1.7	125.8	19.4	2405.3	6.7	132.2	90.8
	0.99	2395.7	4.7	162.4	17.9	2390.3	7.3	158.8	136.0
	0.01	3232.7	0.3	182.6	7.6	2745.3	5.3	131.2	18.9
	0.1	3232.7	0.3	182.6	7.0	2726.7	5.3	131.3	36.0
	0.25	3232.7	0.3	182.6	8.9	2560.0	4.3	133.5	54.4
6_5	0.5	2688.0	1.0	157.1	19.9	2449.0	4.0	138.7	93.4
	0.75	2570.7	1.3	148.2	35.4	2434.0	4.7	142.3	75.2
	0.9	2453.3	2.0	147.5	25.2	2404.7	5.7	155.4	89.9
	0.99	2398.3	4.3	166.3	18.4	2394.3	6.7	175.4	82.9

Table 6.6: Comparison of multi-objective models with RO and ATD for optimally solved Grid instances with K=2

tend to have difficulty handling larger fleet sizes is commonly observed [145, 146], and seen for the MMKWRPP without any aesthetic considerations [119]. It is interesting to note that, while branch-and-cut for model M3 with ATD solves a few more instances to optimality than with RO (155 against 151), the average gaps for the unsolved instances are much greater for the ATD measure. Observe also that the instances seem to be easier for extreme values of α than for the intermediate ones, which are the ones that balance the length of the longest route and the aesthetic measures.

In order to compare the different results obtained with RO and ATD measures on optimal solutions, in Table 6.6 we report the detailed results for the instances for which the optimal solution was found by both models for all the values of α. Columns 3 to 6 give the obtained values for z, RO, and ATD of the optimal solution obtained with the model with RO, as well as the CPU time. Similarly, columns 7 to 10 report the corresponding values for the model with ATD. Note that there seems to be a relation between the values of z and ATD, since both values decrease when α increases in the model with RO (except for the extreme case with $\alpha = 0.99$). This behavior, however, does not seem to occur with the value of RO in the model with ATD. Regarding the values of α, it seems that values 0.75 and 0.9 are preferable to obtain a good balance between z and RO in the first model, while 0.25 and 0.5 could be good options for the second model. We would like to point out that the solutions of both models for the same value of α are, in general, not directly comparable, because of the scale factor contained in the corresponding objective functions. Finally, the computing times are considerably lower for the model with

RO. From here on, we work with model M3, including the RO measure, since we have been more successful computationally with the RO measure.

In order to evaluate the computational performance of the exact algorithm for the model using the RO measure and assess the quality of the heuristic solution procedure, we tested them first on 12 benchmark instances; nine of these are based on larger test instances used in [134] which come from real street networks. For each network, we varied the fleet size between two and four. In addition, we included three test instances generated as the grid instances described in Section 3. The size of these instances ranges from 30 to 83 nodes, with approximately the same number of edges. Problem size, both in terms of network size and fleet size is limited to ensure that the exact procedure was able to solve the problem for a variety of parameter values in under the two-hour threshold allotted, at least in most cases. The test platform was a Macbook Air (2012). The heuristic was coded in Java, running locally with 1 GB of heap space allocated.

In Table 6.7, we compare the exact and the heuristic approaches in terms of z and RO (the values of ATD are also given for information purposes). Since this is a multi-objective optimization, there is no unique optimal solution to the problem. Therefore, we compare the heuristic solution with the solution on the Pareto front that has the next best RO value. For example, if the heuristic produced a solution with an RO value of 5, and there were non-dominated solutions with RO values of 3 and 6, the comparison would be between the heuristic solution and the optimal solution with RO=3. A graphical explanation is given in Figure 6.10. We begin with the heuristic solution indicated by the blue dot. All exact solutions with greater

		Exact				Heuristic				Difference		
Instance	K	z	RO	ATD	Time	z	RO	ATD	Time	Gap z(%)	RO	ATD(%)
Paris	2	2245	2	187.8	2	2245	3	194.1	24	0	1	3.33
Paris	3	1651	15	157.0	235	1651	15	153.2	24	0	0	-2.44
Paris	4	1403	16	127.3	4531	1410	21	132.3	33	0.5	5	3.93
SF	2	1824	2	110.5	5	1824	3	126.0	18	0	1	14.01
SF	3	1243	2	95.0	6	1243	5	101.6	14	0	3	6.92
SF	4	1031	4	92.4	370	1031	7	97.4	13	0	3	5.40
Ist	2	1713	1	137.8	86	1746	5	152.6	30	4.0	4	10.74
Ist	3	1193	8	139.9	1683	1236	13	139.7	41	3.5	5	-0.17
Ist	4	989	20	134.0	7200	1006	23	106.9	54	1.7	3	-20.23
8_8_5	2	4905	2	271.2	7200	5093	9	276.4	110	3.7	7	1.93
7_6_3	3	2012	4	211.0	938	2195	13	226.1	3	8.3	9	7.18
6_5_3	3	1503	6	132.9	78	1618	14	148.9	15	1.5	8	12.04

Table 6.7: Comparison between the solutions provided by the exact and the heuristic procedures

RO values are disregarded. Then, we select the remaining exact solution with the best RO value. Therefore, if the black dots are non-dominated solutions we have computed using the exact procedure, then the one labeled "Closest Exact Solution" would be our point of comparison.

Note that the difference in the min-max objective value of the solutions corresponding to the extreme values of α can be large. If we calculate the deviation between the optimal $\alpha = 0$ solution (only the min-max objective) and the optimal $\alpha = 1$ solution (only the RO objective), the difference is 48.68% averaged over the 12 instances in Table 6.7. This indicates that it can be very costly to improve the visual quality of the routes and that there are non-dominated solutions that would be unacceptable in practice. The deviation in the RO value between the extremes on these 12 instances is, on average, 7.69%, but the effect on the visual quality of the routes depends strongly on the instance.

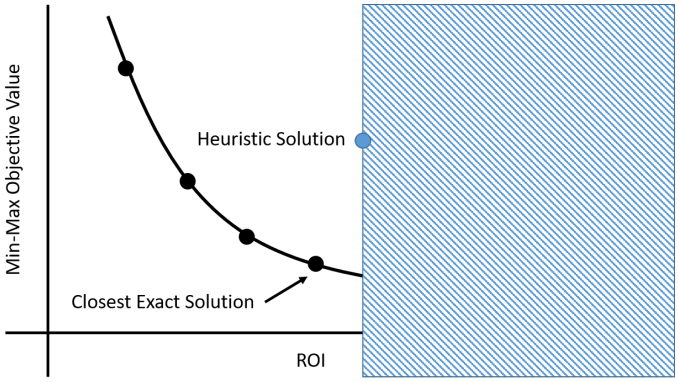


Figure 6.10: The solution on the Pareto front we compare to our heuristic solution.

On average, the heuristic is able to come within 2.07% of the objective value and has a RO 3.8 higher than the closest exact solution. For the instances based on the Paris and San Francisco street networks, we are almost always able to find a solution with the same objective value and a RO that averages about 2.2 higher. These instances have more sparsely connected topologies and, therefore, may be easier for a heuristic procedure to partition well. For the instances based on the Istanbul and artificial street networks, we average 4.03% higher with respect to the min-max objective function and 5.5 higher on RO. The Istanbul instance is quite similar to a grid, and, so, it is not surprising that performance is alike on these instances.

It may initially seem like the solutions produced by the heuristic have poor visual quality, because the RO is always higher (significantly in some cases). However, recall that we defined the "closest" exact solution to be the solution along the Pareto front which has the next lowest RO value compared to the heuristic solution. This means that the heuristic will always produce a set of routes with a higher RO value. Later, in Table 6.8, we compare performance of the heuristic to the $\alpha = 1$

(i.e., pure min-max objective function) optimal solution and see that the heuristic consistently produces routes with lower RO. Our goal is to show that the heuristic is able to strike a balance between the two competing objective functions.

Note that we discuss absolute RO values instead of percentages for two reasons. First, we believe there is a nonlinear relationship between RO and the visual quality of a solution. For example, for medium and large sized instances, a solution with RO of 1 is not twice as visually appealing as a solution with RO of 2; both will be visually appealing solutions. Second, RO values are typically small integers (at least, in the instances we examined). This means that, if we express the difference in RO as a percentage, one additional node of overlap in a small instance will correspond to a much higher percentage than one in a large instance, which does not seem reasonable.

For the two-vehicle instances, the heuristic is slower than the exact procedure, but we can see that with four vehicles the heuristic is much faster. In most cases, the runtime of both solution procedures increases as a function of fleet size. We anticipate this trend to continue for larger fleet sizes. However, the heuristic computed a solution to the San Francisco instance with four vehicles faster than for the same instance with three vehicles. This is because improvement procedures which swap customers between routes occupy the vast majority of the computations for the heuristic ($> 90\%$). Therefore, if these procedures converged very quickly to a local optimum, then this would explain the anomaly.

In addition, we conducted two additional sets of computational experiments to demonstrate the quality and scalability of the solutions produced by the heuristic.

		Exact				Heuristic				Difference		
Instance	K	z	RO	ATD	Time	z	RO	ATD	Time	Gap z(%)	RO	ATD(%)
40_3_6_1	2	4667	10	330.4	53	4896	4	257.2	24	4.90	-6	-22.15
40_3_6_1	3	3265	10	248.8	396	3596	5	208.8	19	10.13	-5	-16.10
40_3_6_1	4	2685	18	243.3	4177	2837	10	198.6	25	5.66	-8	-18.37
40_3_7_3	2	3990	5	349.7	8	4099	4	302.4	8	2.73	-1	-13.53
40_3_7_3	3	2760	2	176.4	6	2851	5	213.8	13	3.29	3	21.20
40_3_7_3	4	2296	10	230.0	359	2511	9	231.1	23	9.36	-1	0.48
40_3_7_5	2	4050	4	313.2	13	4100	1	342.6	8	1.23	-3	9.39
40_3_7_5	3	3007	10	260.0	27	3227	7	217.2	14	7.31	-3	-16.47
40_3_7_5	4	2673	12	216.9	669	2810	7	215.5	25	5.12	-5	-0.67
40_3_8_2	2	4003	9	369.2	18	4042	1	324.4	8	0.97	-8	-12.16
40_3_8_2	3	3007	11	351.7	109	3289	5	348.2	13	9.37	-6	-1.00
40_3_8_2	4	2639	18	342.1	514	2770	5	195.5	20	4.96	-13	-42.83
40_3_8_4	2	4317	9	426.2	30	4700	2	344.5	8	8.87	-7	-19.18
40_3_8_4	3	3153	11	343.1	81	3406	8	297.6	13	8.02	-3	-13.26
40_3_8_4	4	2809	18	280.7	160	2927	9	352.0	20	4.20	-9	25.40
45_3_7_1	2	4179	0	296.0	6	4236	1	315.5	9	1.36	1	6.60
45_3_7_1	3	2921	3	239.1	143	3109	2	241.6	15	6.43	-1	1.05
45_3_7_1	4	2311	9	212.5	1345	2503	8	225.9	23	8.30	-1	6.31
45_3_7_5	2	4295	8	360.6	27	4535	1	320.4	8	5.58	-7	-11.15
45_3_7_5	3	3222	13	261.0	112	3314	5	265.0	14	2.85	-8	1.50
45_3_7_5	4	2708	20	336.2	2073	2900	9	223.2	35	7.09	-11	-33.61
45_3_8_2	2	3726	5	306.2	20	3766	0	260.1	10	1.07	-5	-15.07
45_3_8_2	3	2758	12	282.1	61	2951	6	314.3	18	6.99	-6	11.39
45_3_8_2	4	2461	17	271.9	3139	2617	9	230.8	22	6.33	-8	-15.13
45_3_8_3	2	3837	8	294.0	27	3870	3	303.4	10	0.86	-5	3.21
45_3_8_3	3	2754	8	276.0	3619	2783	2	179.8	16	1.05	-6	-34.84
45_3_8_3	4	2141	11	152.7	4309	2314	6	195.5	22	8.08	-5	27.97
45_3_8_4	2	4163	0	265.9	227	4163	0	282.9	8	0.00	0	6.39
45_3_8_4	3	2937	14	359.9	189	3228	9	300.7	16	9.90	-5	-16.46
45_3_8_4	4	2443	15	270.9	7200	2610	3	199.5	21	6.83	-12	-26.36
50_3_7_2	2	4731	5	298.3	7	4960	4	297.2	9	4.84	-1	-0.39
50_3_7_2	3	3385	8	253.6	7200	3551	7	232.1	17	4.90	-1	-8.46
50_3_7_2	4	2704	15	277.5	7200	2925	16	179.0	25	8.17	1	-35.49
50_3_7_4	2	3954	5	303.9	10	4131	2	260.0	11	4.47	-3	-14.45
50_3_7_4	3	2868	12	295.8	937	3055	2	227.5	17	6.52	-10	-23.11
50_3_7_4	4	2390	20	245.0	4371	2749	6	292.3	25	15.02	-14	19.31

Table 6.8: Comparison between the solutions provided by the exact and the heuristic procedures

The first set of experiments involved a set of 20 slightly larger street networks containing between 100 and 200 edges (and between 40 and 60 required edges).

		Exact				Heuristic				Difference		
Instance	K	z	RO	ATD	Time	z	RO	ATD	Time	Gap z(%)	RO	ATD(%)
50_3_8_5	2	4047	4	384.6	13	4181	2	350.5	10	3.31	-2	-8.85
50_3_8_5	3	3066	10	247.5	2179	3108	3	224.0	15	1.36	-7	-9.53
50_3_8_5	4	2457	10	225.9	6840	2558	7	245.7	22	4.11	-3	8.78
50_3_9_1	2	4679	7	368.3	18	5017	2	355.5	9	7.22	-5	-3.47
50_3_9_1	3	3459	11	341.7	833	3715	5	257.0	16	7.40	-6	-24.77
50_3_9_1	4	2846	15	254.2	4313	3016	6	262.2	22	5.97	-9	3.14
50_3_9_3	2	4059	9	408.6	25	4598	4	333.0	11	13.27	-5	-18.49
50_3_9_3	3	3111	12	266.2	4992	3453	6	259.2	18	10.99	-6	-2.62
50_3_9_3	4	2690	16	253.4	6636	2878	12	249.0	22	6.98	-4	-1.72
55_3_7_5	2	4359	2	334.9	17	4405	2	272.2	10	1.05	0	-18.71
55_3_7_5	3	3102	15	266.1	2171	3433	6	254.7	18	10.67	-9	-4.29
55_3_7_5	4	2519	15	309.6	3807	2827	7	184.4	24	12.22	-8	-40.44
55_3_8_1	2	4803	6	326.9	59	5050	4	297.5	12	5.14	-2	-9.01
55_3_8_1	3	3358	10	292.5	7200	3471	5	263.9	22	3.36	-5	-9.77
55_3_8_1	4	2692	14	232.6	7200	2893	9	205.9	24	7.46	-5	-11.46
55_3_8_2	2	4249	5	320.6	121	4691	4	260.8	12	10.40	-1	-18.66
55_3_8_2	3	3036	11	298.0	577	3396	9	234.4	24	11.85	-2	-21.34
55_3_8_2	4	2488	14	271.1	7200	2816	9	220.6	27	13.18	-5	-18.61
55_3_9_4	2	4082	11	291.6	75	4284	3	334.8	12	4.94	-8	14.83
55_3_9_4	3	3060	12	284.6	798	3445	5	325.8	20	12.58	-7	14.46
55_3_9_4	4	2651	19	258.2	2982	2893	10	256.4	30	9.12	-9	-0.68
55_3_10_3	2	4746	7	340.5	80	5021	4	307.4	11	5.79	-3	-9.71
55_3_10_3	3	3406	7	315.5	7200	3687	7	268.4	17	8.25	0	-14.92
55_3_10_3	4	2840	21	343.5	7200	3120	8	189.3	25	9.85	-13	-44.88

Table 6.8: Comparison between the solutions provided by the exact and the heuristic procedures (continued).

While the exact approach that takes RO into account would take a prohibitively long amount of time to run, we can use an exact approach from [144] to solve the problem with a purely min-max objective function and compare the results. For each of the 20 street networks, we solved the problem with fleet size varying between two and four vehicles for a total of 60 test instances. The results are presented in Table 6.8. On average, the heuristic produces a solution with an objective value that is 6.5% above the optimal value, or the upper bound found by the exact procedure after 7200 seconds. However, these solutions have an RO value that is 5.1 lower

on average than the solutions produced by the exact procedure. To put this into context, the heuristic produces solutions that have approximately half as much overlap, on average.

The second set of experiments involved a set of 15 large street networks containing between 300 and 620 edges (and between 100 and 400 required edges). We vary the fleet size between two and five for these networks, for a total of 60 instances. Neither exact approach is capable of solving these instances within several hours, so these experiments are intended only to demonstrate the ability of the heuristic to solve instances of this size. The results are contained in Table 6.9. Again, in some cases, run time decreases with increased fleet size. As before, we believe this is due to the improvement procedures converging quickly on a local optimum. For the smallest instances of this set, the heuristic is capable of finishing in under a minute, while for the most complex instances (i.e., the instances with the largest number of required edges and vehicles), it requires over 40 minutes. The time required is approximately quadratic in the number of required edges. Since the improvement phase of the heuristic only considers moves between required edges, the total number of edges in the network has relatively little effect on the run time. For example, the 15_10_5_1 network has 358 edges while the 200_3_24_1 network has 615 edges. However, both instances have approximately the same number of required edges, (180 and 185, respectively), and so they require approximately the same amount of time to solve. This reinforces the observation that the improvement phase dominates the run time of the heuristic.

6.6 Conclusions

In this chapter, we developed and compared several IP formulations for solving the Aesthetic MMKWRPP. We have studied two aesthetic measures, RO and ATD, and we have tried three different models that incorporate these measures. The best results are obtained by including the aesthetic measures in a multi-objective function. When comparing the solutions obtained with RO and ATD visually, RO seems to provide "nicer" solutions. Several random instances have been generated and a branch-and-cut algorithm has been tested to solve the multi-objective problem with the aesthetic measures. Finally, we extended an existing cluster first, route second heuristic for the MMKWRPP and showed it to be competitive with solutions on the Pareto front. We demonstrated that this modified heuristic is capable of scaling to problem instances with hundreds of nodes.

In our heuristic, we found that an effective perturbation strategy for retaining the visual appeal of a set of routes was to operate directly on the center points of the routes and to reassign customers based on the proximity of these center points. While we identified a simple way of incorporating these into a heuristic that was sufficient for our set of test instances, it would be interesting to see this approach used in the context of a more sophisticated metaheuristic and validated on larger test instances.

In addition, while we can see that optimizing for RO alone can often lead to a solution that is very unbalanced, in many cases it's not clear which solution on the Pareto front is the best or most preferable. It is also not clear whether or not all

of them are consistently acceptable in the opinion of typical distribution managers. If it is possible to gain further insight into the considerations they use to determine the overall quality of a route, it may be possible to narrow the search.

Instance	$\|V\|$	$\|E\|$	$\|E_R\|$	K	z	RO	Time(scs)
100_3_8_1	100	311	105	2	6912	2	46
100_3_8_1	100	311	105	3	5177	17	54
100_3_8_1	100	311	105	4	4107	17	55
100_3_8_1	100	311	105	5	3566	18	74
100_3_8_2	100	310	111	2	7234	3	40
100_3_8_2	100	310	111	3	5150	7	46
100_3_8_2	100	310	111	4	4078	16	62
100_3_8_2	100	310	111	5	3427	15	60
150_3_11_2	150	467	160	2	9408	6	93
150_3_11_2	150	467	160	3	6443	12	137
150_3_11_2	150	467	160	4	5085	18	164
150_3_11_2	150	467	160	5	4505	27	189
150_3_7_1	150	465	177	2	9403	5	132
150_3_7_1	150	465	177	3	6675	14	132
150_3_7_1	150	465	177	4	5283	14	177
150_3_7_1	150	465	177	5	4482	25	219
15_10_3_1	150	358	136	2	7430	3	61
15_10_3_1	150	358	136	3	5381	10	67
15_10_3_1	150	358	136	4	4323	17	97
15_10_3_1	150	358	136	5	3576	13	135
15_10_5_1	150	358	180	2	8946	5	95
15_10_5_1	150	358	180	3	6105	10	156
15_10_5_1	150	358	180	4	4957	22	175
15_10_5_1	150	358	180	5	4217	22	130
15_10_7_1	150	358	258	2	11821	17	424
15_10_7_1	150	358	258	3	8564	19	324
15_10_7_1	150	358	258	4	6518	30	364
15_10_7_1	150	358	258	5	5409	37	716
15_15_3_1	225	550	204	2	11266	7	199
15_15_3_1	225	550	204	3	8225	12	277
15_15_3_1	225	550	204	4	6764	19	482
15_15_3_1	225	550	204	5	5810	19	473
15_15_5_1	225	550	270	2	13062	6	349
15_15_5_1	225	550	270	3	9028	13	372
15_15_5_1	225	550	270	4	6906	14	751
15_15_5_1	225	550	270	5	5771	30	732
15_15_7_1	225	550	396	2	17817	21	1406
15_15_7_1	225	550	396	3	12591	36	970
15_15_7_1	225	550	396	4	9854	33	2726
15_15_7_1	225	550	396	5	8066	54	2618

Table 6.9: Results of the heuristic on a set of larger instances.

Instance	$\|V\|$	$\|E\|$	$\|E_R\|$	K	z	RO	Time(scs)
200_3_16_2	200	616	207	2	9792	7	168
200_3_16_2	200	616	207	3	6939	15	250
200_3_16_2	200	616	207	4	5271	15	246
200_3_16_2	200	616	207	5	4449	26	362
200_3_24_1	200	615	185	2	9659	3	138
200_3_24_1	200	615	185	3	6693	5	149
200_3_24_1	200	615	185	4	5308	12	189
200_3_24_1	200	615	185	5	4462	27	206
20_10_3_1	200	484	178	2	7610	6	129
20_10_3_1	200	484	178	3	5406	5	160
20_10_3_1	200	484	178	4	4368	11	294
20_10_3_1	200	484	178	5	3708	14	261
20_10_5_1	200	484	247	2	8790	5	258
20_10_5_1	200	484	247	3	6167	11	315
20_10_5_1	200	484	247	4	4664	21	596
20_10_5_1	200	484	247	5	3896	23	486
20_10_7_1	200	484	353	2	11698	8	588
20_10_7_1	200	484	353	3	7999	16	1145
20_10_7_1	200	484	353	4	6180	37	1452
20_10_7_1	200	484	353	5	5120	36	2245

Table 6.9: Results of the heuristic on a set of larger instances (continued).

Chapter 7: Conclusions

In this dissertation, we developed two software tools for solving and generating arc routing problems. The first tool (OAR Lib) is a library of solvers and heuristics for standard arc routing problems that can be used as a standalone system, or can be used to develop algorithms for more complex problems. The second tool (OAR Bench) is a GIS tool that enables users to quickly generate test instances that accurately reflect the structural properties of real-world street networks. OAR Lib and OAR Bench are available for use by the research community. In addition, the source code is open for both tools.

We modeled and developed heuristics for two arc routing problem variants. The first variant is the Windy Rural Postman Problem with Zigzag Time Windows. Here, a time-constrained secondary service mode was used to serve customers on both sides of the street. The second variant is the Min-Max K Windy Rural Postman Problem. In this variant, we constructed routes for a set of K vehicles, where the objective function was the cost of the longest route. This objective measured route balance as well as cost which, in practice, reduces unbalanced driving assignments. For both variants, we developed simple, scalable heuristics that provided comparable solution quality to existing procedures.

Finally, we used the Min-Max K Windy Rural Postman Problem to investigate alternative measures of route quality that take into account the aesthetic properties of the vehicle routes. In practice, properties such as compactness and overlap are important to assess the quality of proposed routes. For example, non-overlapping routes allow drivers to become familiar with the streets in a particular region and allow them to handle last-minute changes to customer stops on a route. We introduced overlapping convex hulls to measure the degree of route overlap. Our heuristic for the MMKWRPP performed favorably with respect to these metrics compared to an existing procedure. In addition, we considered a multiobjective formulation of the problem and showed that our heuristic produced solutions that were comparable to optimal solutions.

Appendices

Appendix A: Partial route formation example

We provide an example that shows how the initial partial routes are formed during the initialization phase of our heuristic. In Figure A.1, we show the street network, with three zigzag optional street segments that must be zigzagged before time 40. Dotted single edges do not require service. Dotted double edges with two arrows have a zigzag option. In this instance, all edges with a zigzag option have a time window shown by the clock that begins at time 0 and ends at time 40. The traversal cost, service cost, and zigzag cost are c_{ij}, s_{ij}, and z_{ij}, respectively. In Figure A.2, we apply our heuristic to the instance given in Figure A.1. In this case, segment $(3, 4)$ is selected as the seed customer. Segment $(7, 8)$ is the first customer to be selected for insertion based on the PFIH score. We select the street with the lowest score to insert. However, this insertion would cause the seed customer to be visited outside of its time window, so the insertion is not made. After the insertion in Figure A.2 fails, segment $(0, 1)$ is considered for insertion. In Figure A.3, we show its successful insertion. The route that results has a cost of $6+4+5+3+3+8 = 29 < 40$, so the partial route is feasible, and the insertion is made. In Figure A.4, we show the final partial route $0 - (\text{zigzag}) - 1 - 2 - 3 - (\text{zigzag}) - 4$.

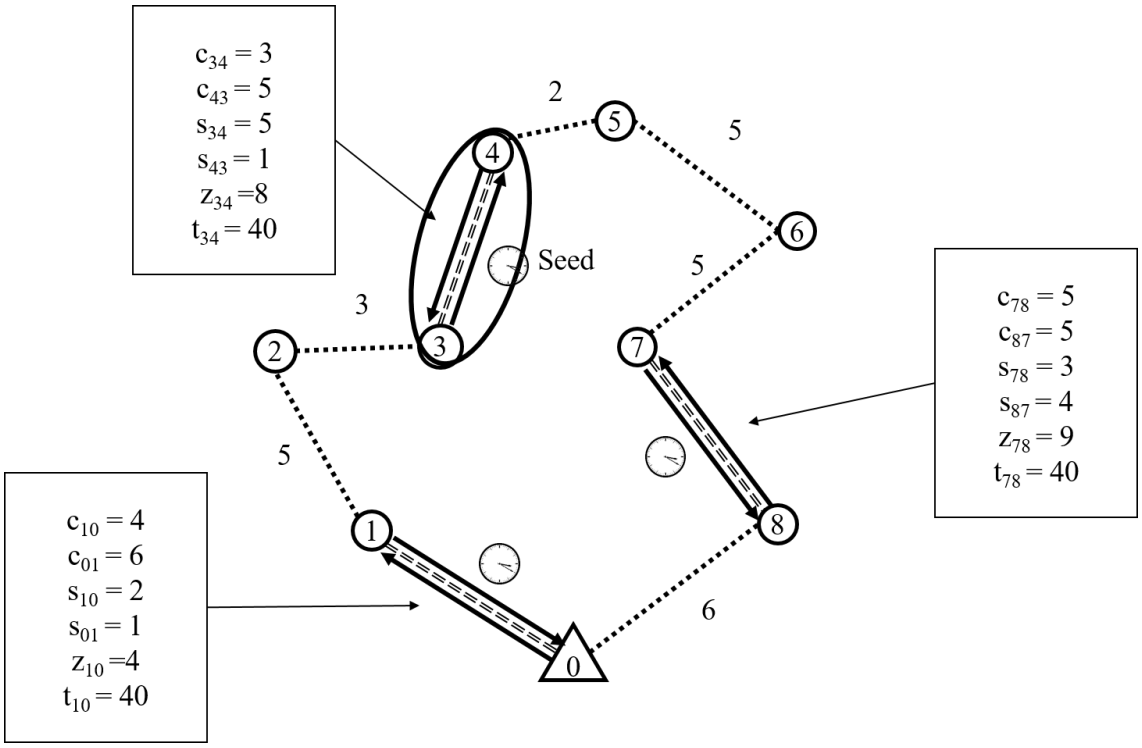


Figure A.1: A WRPPZTW instance.

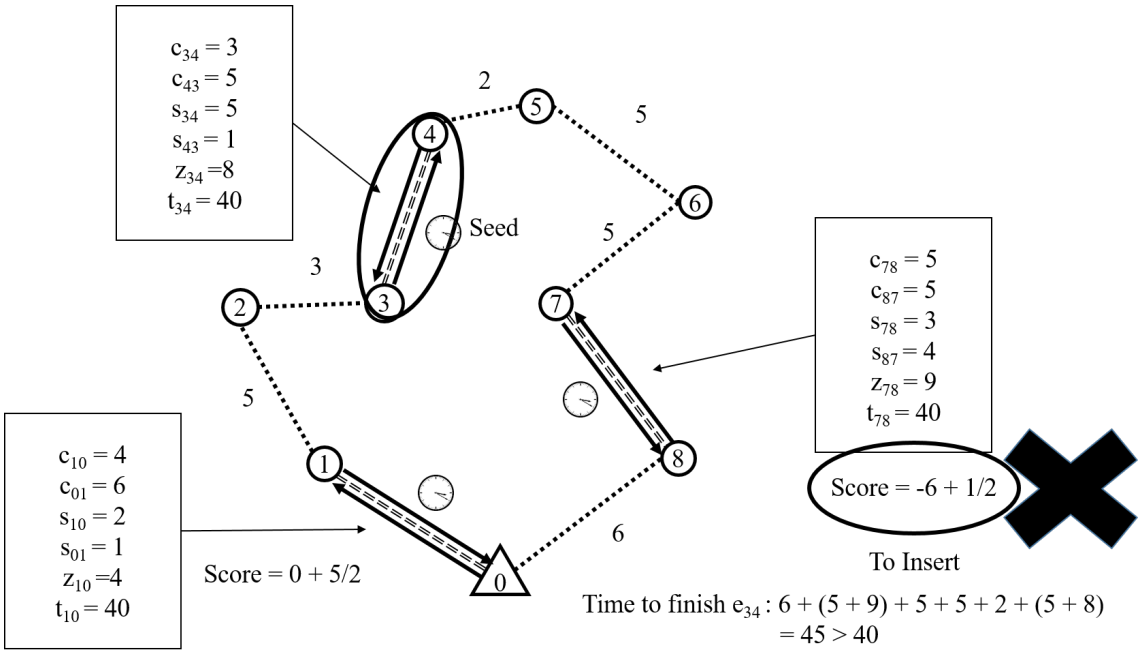


Figure A.2: First attempted insertion by our heuristic fails.

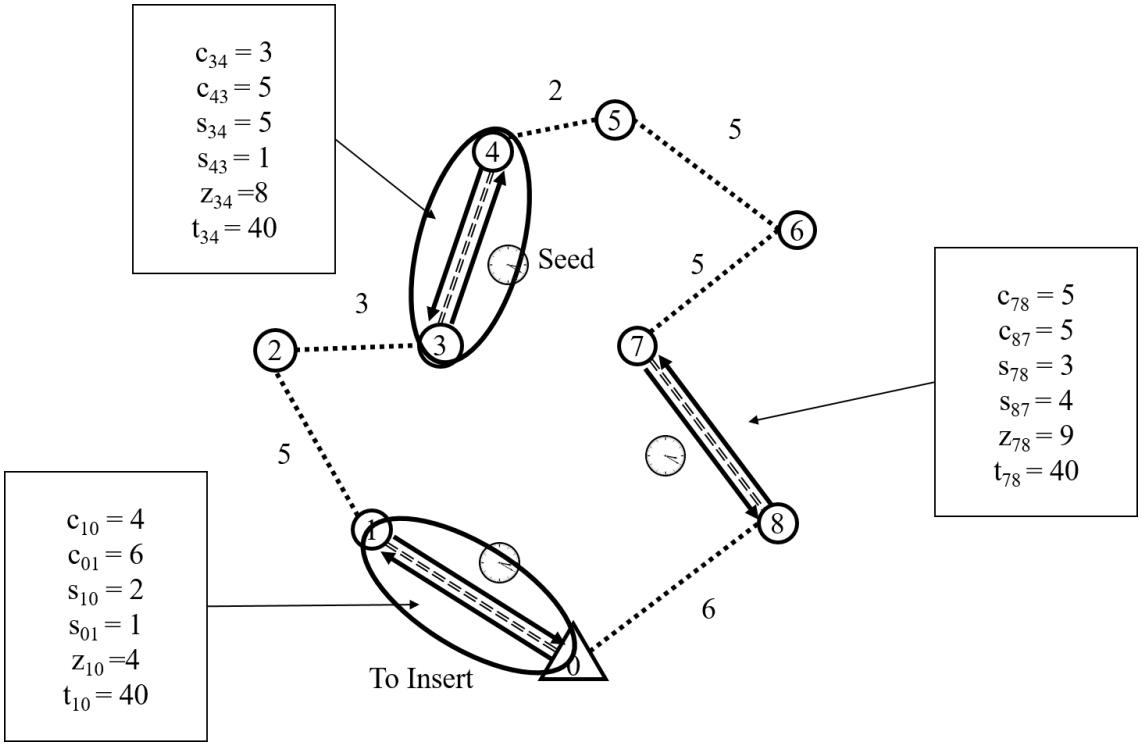


Figure A.3: Second attempted insertion.

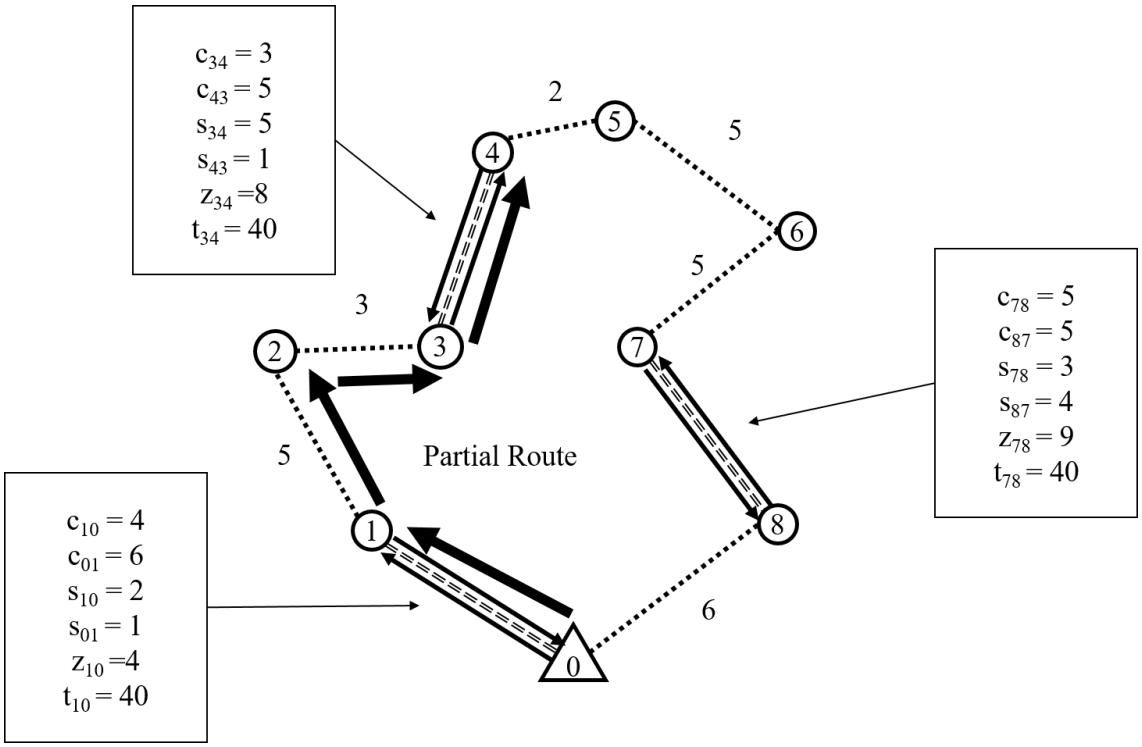


Figure A.4: Final partial route.

Appendix B: Deadhead Preprocessor

Algorithm 10 Preprocessing function deadhead

Input: The graph $G = (V, E)$, and $E_R \subseteq E$.

Output: A set of edges $E_{dh} \subseteq E_R$ that must be deadheaded in the final solution.

```
1: continue = true
2: Initialize E_dh = {}
3: while continue do
4:     E_check = E_R \ E_dh
5:     continue = false
6:     for e = (i, j) in E_check do
7:         if degree(i) = 1 OR degree(j) = 1 then
8:             add e to E_dh
9:             continue = true
10:            remove e from the graph G
11:        end if
12:    end for
13: end while
```

Appendix C: Detailed Example Using OAR Bench

In this appendix, we show how to use OAR Bench to generate an instance. Suppose we want to generate an instance based on a portion of the street network of the city of Amsterdam. In Figure C.1, we show the screen when OAR Bench is first opened. The search bar on the map is used to find Amsterdam. The result of the search is shown in Figure C.2. We then zoom in so that the portion of the street network that we want is onscreen. In Figure C.3, after clicking the Estimate Instance Size button, the streets found in the OSM database are drawn on the map in red. The green notification box displays how many edges were returned. After clicking the Generate Instance button, the street network is saved to a file. In Figure C.4, we switch the Display tab and load the network. This network has 1034 nodes and 1396 edges. We use the Auto Trim procedure to delete disconnected nodes and edges. The network after Auto Trim is shown in Figure C.5; it has 992 nodes and 1353 edges. Notice that a cluster of streets in the top right have been eliminated from the network. Finally, we use the Randomize button to set some of the streets as required. In Figure C.6, the interface that specifies the probability that an edge is required depending on its priority is shown. The final instance is shown in Figure C.7. The final instance can be exported as a text file (Figure C.8) that can be parsed by a solver and exchanged freely among researchers.

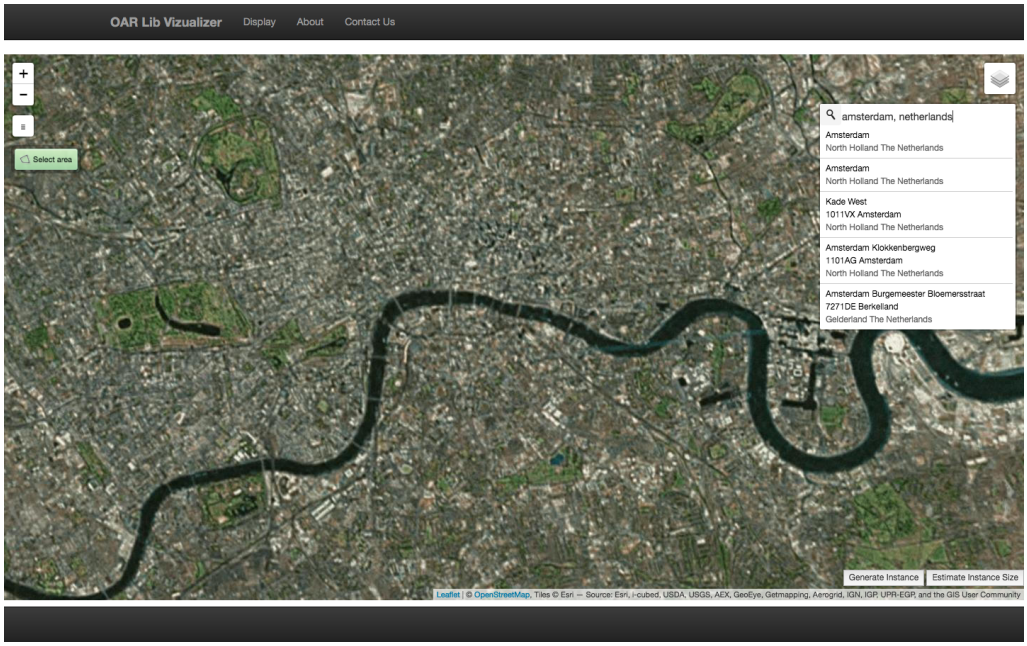


Figure C.1: Map search

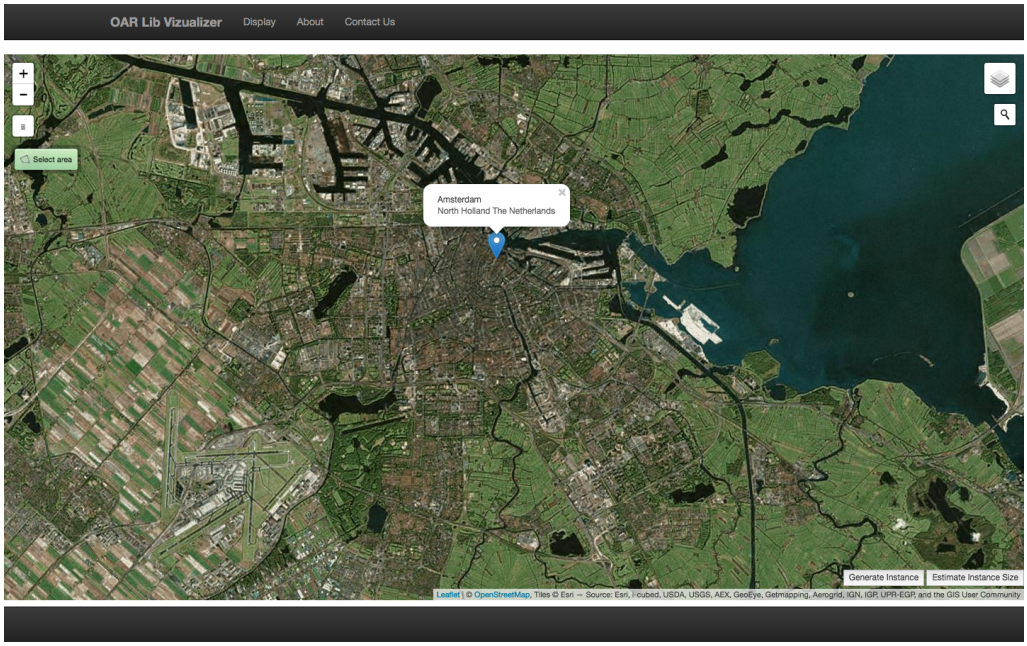


Figure C.2: Map centered on the city of Amsterdam

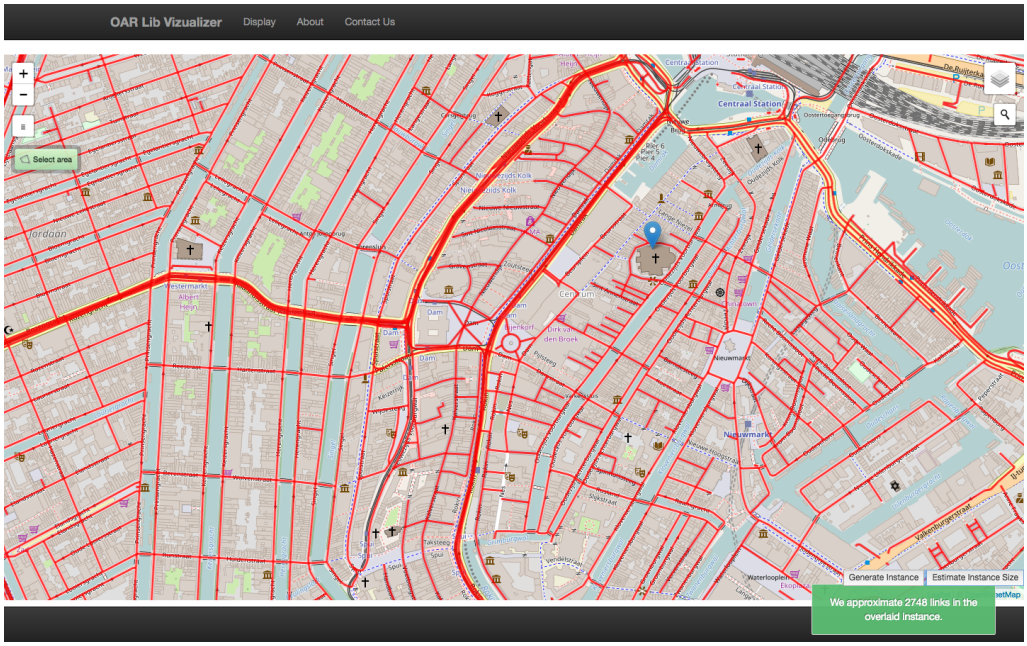


Figure C.3: Instance estimated

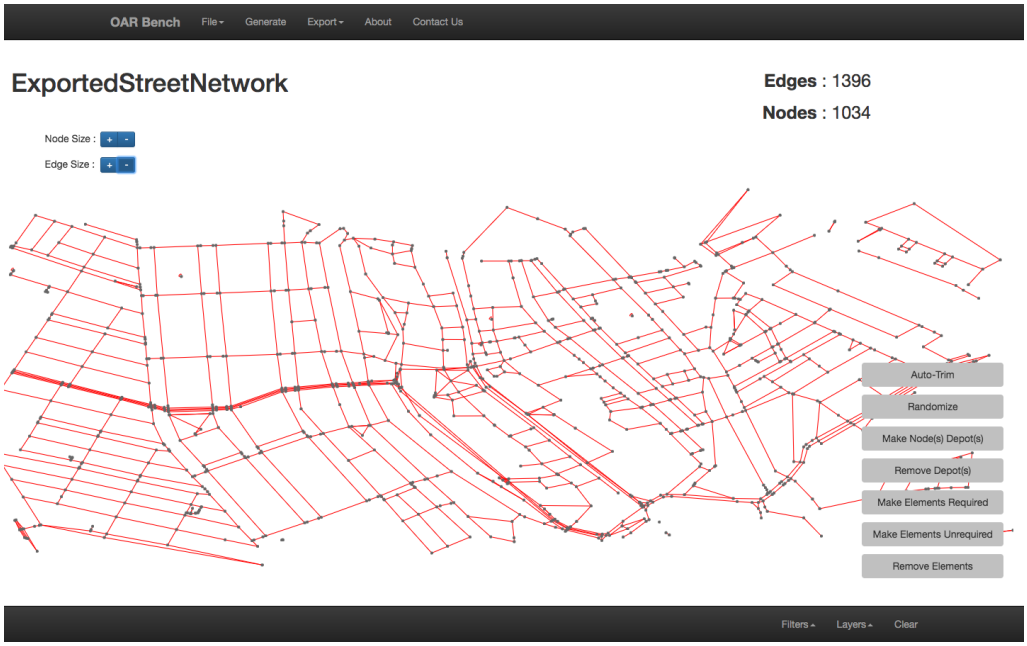


Figure C.4: Importing the Amsterdam instance

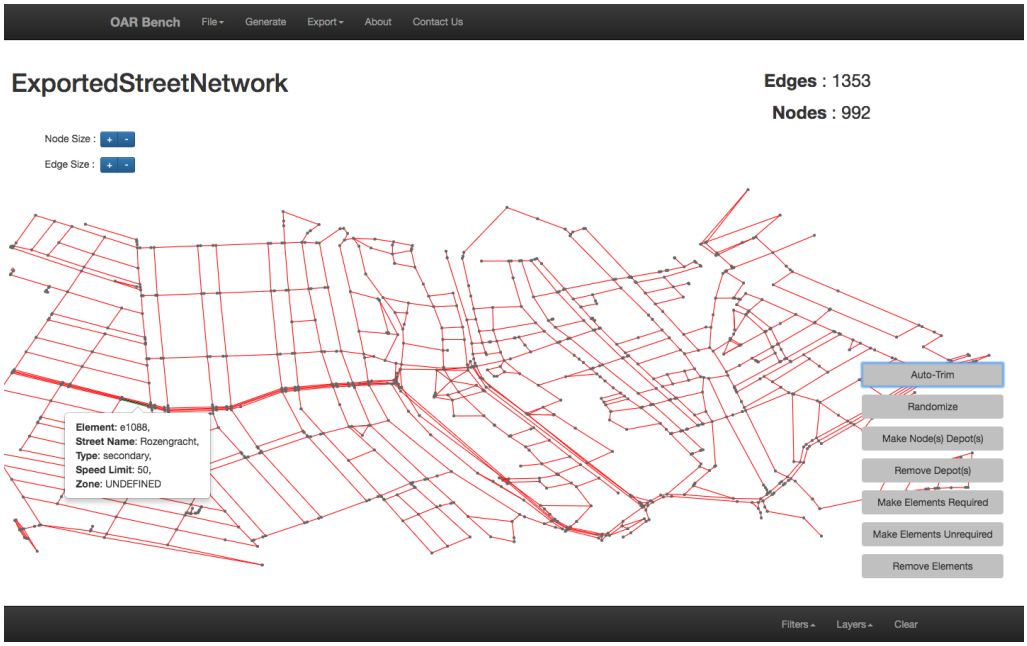


Figure C.5: Network after applying the Auto Trim procedure

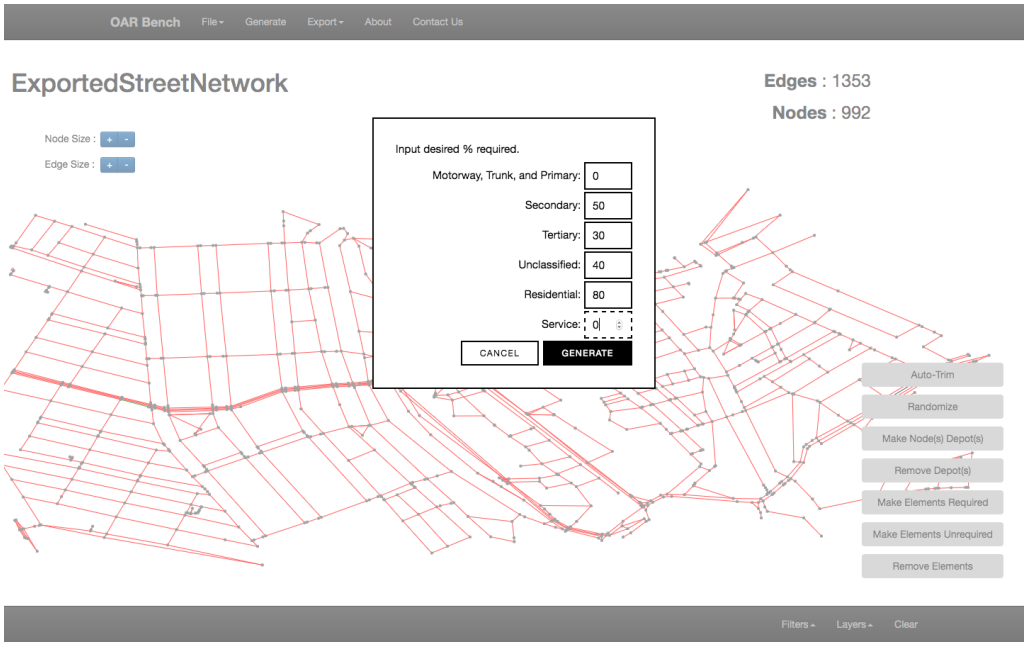


Figure C.6: Randomizing required streets by type of street

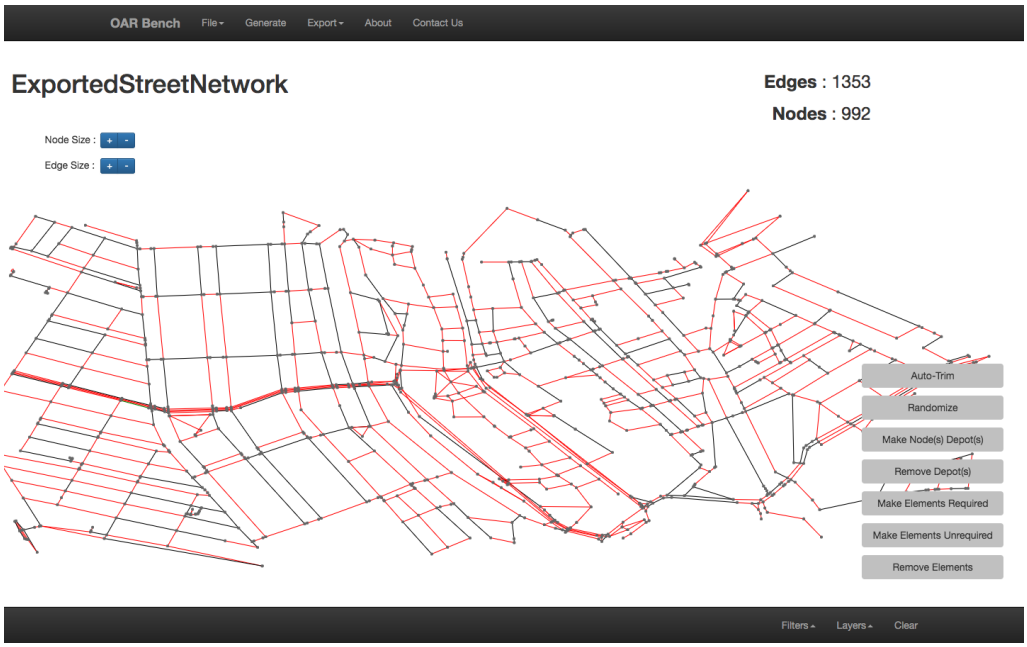


Figure C.7: Amsterdam instance after randomization. Black edges are required; red edges are not required.

```
1024, 12476, 4518
1025, 12422, 4513
1026, 15980, 9096
1027, 16255, 9270
1028, 16237, 9282
1029, 1738, 3086
1030, 1671, 3172
1031, 615, 2566
1032, 549, 2652
1033, 6761, 9536
1034, 6677, 9447
================================
EDGES: ID, V1, V2, DIST, REQUIRED, DIRECTED, TYPE, NAME, MAX_SPEED, ZONE
================================
e1363, 1016, 288, 39.44616584663204, true, true, unclassified, Tweede Leliedwarsstraat, NaN, UNDEFINED
e1368, 1019, 1017, 96.25487000666512, false, false, pedestrian, Beurspassage, NaN, UNDEFINED
e1369, 282, 1018, 46.84015371452148, false, false, pedestrian, Beurspassage, NaN, UNDEFINED
e1370, 1018, 1019, 671.3784327784144, false, false, pedestrian, Beurspassage, NaN, UNDEFINED
e1324, 1002, 1003, 531.6483800407935, false, false, pedestrian, undefined, NaN, UNDEFINED
e1322, 1000, 1001, 369.6281915655244, false, false, pedestrian, undefined, NaN, UNDEFINED
e1337, 992, 877, 67.7421582177598, true, true, secondary, Nieuwezijds Voorburgwal, 50, UNDEFINED
e1359, 1010, 1009, 53.009433122794285, false, true, unclassified, Prins Hendrikkade, NaN, UNDEFINED
e1320, 1005, 1002, 574.5441671447027, false, false, pedestrian, Keizersstraat, NaN, UNDEFINED
e1313, 997, 998, 142.41137595009747, true, false, unclassified, undefined, NaN, UNDEFINED
e1312, 998, 694, 23.600847442411894, true, false, unclassified, undefined, NaN, UNDEFINED
e1326, 104, 711, 619.7757659024754, false, false, pedestrian, J.W. Siebbeleshof, NaN, UNDEFINED
```

Figure C.8: Text file of the Amsterdam instance

Bibliography

[1] Euler, Leonhard. (1741). Solutio problematis ad geometriam situs pertinentis. *Commentarii academiae scientiarum Petropolitanae*, 8, 128-140.

[2] Guan, M. (1962). Graphical Programming Using Odd and Even Points. *Chinese Math*, 1, 273-277.

[3] Eiselt, H. A., Gendreau, M., and Laporte, G. (1995). Arc routing problems, part I: The Chinese postman problem. *Operations Research*, 43(2), 231-242.

[4] Thimbleby, H. (2003). The directed chinese postman problem. *Software Practice and Experience*, 33(11), 1081-1096.

[5] Edmonds, J., and Johnson, E. L. (1973). Matching, Euler tours and the Chinese postman. *Mathematical Programming*, 5(1), 88-124.

[6] Frederickson, G. N. (1979). Approximation algorithms for some postman problems. *Journal of the ACM*, 26(3), 538-554.

[7] Yaoyuenyong, K., Charnsethikul, P., and Chankong, V. (2002). A heuristic algorithm for the mixed Chinese postman problem. *Optimization and Engineering*, 3(2), 157-187.

[8] Win, Z. (1989). On the windy postman problem on Eulerian graphs. *Mathematical Programming*, 44(1-3), 97-112.

[9] Grötschel, M., and Win, Z. (1992). A cutting plane algorithm for the windy postman problem. *Mathematical Programming*, 55(1-3), 339-358.

[10] Campos, V., and Savall, J. V. (1995). A computational study of several heuristics for the DRPP. *Computational Optimization and Applications*, 4(1), 67-77.

[11] Benavent, E., Corberán, A., Piñana, E., Plana, I., and Sanchis, J. M. (2005). New heuristic algorithms for the windy rural postman problem. *Computers & Operations Research*, 32(12), 3111-3128.

[12] Groër, C., Golden, B., and Wasil, E. (2010). A library of local search heuristics for the vehicle routing problem. *Mathematical Programming Computation*, 2(2), 79-101.

[13] Hierholzer, C., and Wiener, C. (1873). Über die Möglichkeit, einen Linienzug ohne Wiederholung und ohne Unterbrechung zu umfahren. *Mathematische Annalen*, 6(1), 30-32.

[14] Bastian, M., Heymann, S., and Jacomy, M. (2009). Gephi: an open source software for exploring and manipulating networks. *ICWSM*, 8, 361-362.

[15] M. Haklay, and P. Weber, Openstreetmap: User-generated street maps, *IEEE* Pervasive Computing 7:4 (2008), 12-18.

[16] Floyd, R. W. (1962). Algorithm 97: shortest path. *Communications of the ACM*, 5(6), 345.

[17] Dijkstra, E. W. (1959). A note on two problems in connexion with graphs. *Numerische Mathematik*, 1(1), 269-271.

[18] Minimum cost flow tutorial. http://community.topcoder.com/tc?module=Static&d1=tutorials&d2=minimumCostFlow2

[19] Kolmogorov, V. (2009). Blossom V: A new implementation of a minimum cost perfect matching algorithm. *Mathematical Programming Computation*, 1(1), 43-67.

[20] Prim, R. C. (1957). Shortest connection networks and some generalizations. *Bell System Technical Journal*, 36(6), 1389-1401.

[21] Lau, H.T. *A Java library of graph algorithms and optimization*. CRC Press, 2010.

[22] GNU Trove library. http://trove.starlight-systems.com/

[23] Eiselt, H. A., Gendreau, M., and Laporte, G. (1995). Arc routing problems, part II: The rural postman problem. *Operations Research*, 43(3), 399-414.

[24] Arc Routing Problems: Data Instances http://www.uv.es/corberan/instancias.htm

[25] Tahchiev, P., Leme, F., Massol, V., and Gregory, G. (2010). *JUnit in action*. Manning Publications Co..

[26] A. A. Assad, Leonhard Euler: A brief appreciation, Networks 49 (3) (2007) 190–198 (2007).

[27] I. Gribkovskaia, Ø. Halskau, G. Laporte, The bridges of Königsberg, a historical perspective, Networks 49 (3) (2007) 199–203 (2007).

[28] Á. Corberán, G. Laporte, Arc routing: problems, methods, and applications, SIAM, 2013.

[29] A. Corberán, C. Prins, Recent results on arc routing problems: An annotated bibliography, Networks 56 (1) (2010) 50–69 (2010).

[30] M. Mourão, L. Pinto, An updated annotated bibliography on arc routing problems (under review 2017).

[31] Y. Zhang, Y. Mei, K. Tang, K. Jiang, Memetic algorithm with route decomposing for periodic capacitated arc routing problem, Applied Soft Computing 52 (2017) 1130–1142 (2017).

[32] R. Shang, K. Dai, L. Jiao, R. Stolkin, Improved memetic algorithm based on route distance grouping for multiobjective large scale capacitated arc routing problems, IEEE Transactions on Cybernetics 46 (4) (2016) 1000–1013 (2016).

[33] J. Wang, K. Tang, J. A. Lozano, X. Yao, Estimation of the distribution algorithm with a stochastic local search for uncertain capacitated arc routing problems, IEEE Transactions on Evolutionary Computation 20 (1) (2016) 96–109 (2016).

[34] E. J. Willemse, J. W. Joubert, Splitting procedures for the mixed capacitated arc routing problem under time restrictions with intermediate facilities, Operations Research Letters 44 (5) (2016) 569–574 (2016).

[35] E. Fernández, D. Fontana, M. G. Speranza, On the collaboration uncapacitated arc routing problem, Computers & Operations Research 67 (2016) 120–131 (2016).

[36] G. A. Zeni, M. Menzori, P. Martins, L. A. Meira, Vrpbench: A vehicle routing benchmark tool, arXiv preprint arXiv:1610.05402 (2016).

[37] M. Haklay, P. Weber, Open street map: User-generated street maps, IEEE Pervasive Computing 7 (4) (2008) 12–18 (2008).

[38] J. Anderson, R. Soden, K. M. Anderson, M. Kogan, L. Palen, Epic-osm: A software framework for open street map data analytics, in: T. X. Bui, J. Ralph H. Sprague (Eds.), System Sciences (HICSS), 2016 49th Hawaii International Conference on System Sciences, IEEE, Koloa, HI, USA, 2016, pp. 5468–5477.

[39] P. Yadav, S. Deshpande, R. Sengupta, Animating maps: Visual analytics meets geoweb 2.0, in: C. Y. Griffith D., D. D. (Eds.), Advances in Geocomputation, Springer, Cham, 2017, pp. 75–84.

[40] M. Göbelbecker, C. Dornhege, et al., Realistic cities in simulated environments-an open street map to robocup rescue converter, in: Fourth International Workshop on Synthetic Simulation and Robotics to Mitigate Earthquake Disaster (SRMED 2009), Graz, Austria, 2009.

[41] L. Palen, R. Soden, T. J. Anderson, M. Barrenechea, Success & scale in a data-producing organization: The socio-technical evolution of open street map in response to humanitarian events, in: B. Begole, J. Kim (Eds.), Proceedings of the 33rd annual ACM conference on human factors in computing systems, ACM, Seoul, Korea, 2015, pp. 4113–4122.

[42] L. Kiilerich, S. Wøhlk, New large-scale data instances for carp and new variations of carp, INFOR: Information Systems and Operational Research (2017) 1–32 (2017).

[43] M. Constantino, L. Gouveia, M. C. Mourão, A. C. Nunes, The mixed capacitated arc routing problem with non-overlapping routes, European Journal of Operational Research 244 (2) (2015) 445–456 (2015).

[44] E. J. Willemse, J. W. Joubert, Benchmark dataset for undirected and mixed capacitated arc routing problems under time restrictions with intermediate facilities, Data in Brief 8 (2016) 972–977 (2016).

[45] E. Benavent, A. Carrotta, A. Corberán, J. M. Sanchis, D. Vigo, Lower bounds and heuristics for the windy rural postman problem, European Journal of Operational Research 176 (2) (2007) 855–869 (2007).

[46] N. Maxemchuk, Routing in the manhattan street network, IEEE Transactions on Communications 35 (5) (1987) 503–512 (1987).

[47] P. Erdos, A. Rényi, On the evolution of random graphs, Publ. Math. Inst. Hung. Acad. Sci 5 (1) (1960) 17–60 (1960).

[48] A.-L. Barabási, R. Albert, Emergence of scaling in random networks, Science 286 (5439) (1999) 509–512 (1999).

[49] M. M. Solomon, Algorithms for the vehicle routing and scheduling problems with time window constraints, Operations Research 35 (2) (1987) 254–265 (1987).

[50] W. L. Pearn, T. Wu, Algorithms for the rural postman problem, Computers & Operations Research 22 (8) (1995) 819–828 (1995).

[51] M. Kiuchi, Y. Shinano, R. Hirabayashi, Y. Saruwatari, An exact algorithm for the capacitated arc routing problem using parallel branch and bound method, in: Spring National Conference of the Operational Research Society of Japan, 1995, pp. 28–29.

[52] Belenguer, José-Manuel, et al. "Lower and upper bounds for the mixed capacitated arc routing problem." Computers & Operations Research 33.12 (2006): 3363-3383.

[53] B. L. Golden, J. S. DeArmon, E. K. Baker, Computational experiments with algorithms for a class of routing problems, Computers & Operations Research 10 (1) (1983) 47–59 (1983).

[54] L. Y. Li, R. W. Eglese, An interactive algorithm for vehicle routeing for winter–gritting, Journal of the Operational Research Society 47 (2) (1996) 217–228 (1996).

[55] Brandão, José, and Richard Eglese. "A deterministic tabu search algorithm for the capacitated arc routing problem." Computers & Operations Research 35.4 (2008): 1112-1126.

[56] Beullens, Patrick, Luc Muyldermans, Dirk Cattrysse, and Dirk Van Oudheusden. "A guided local search heuristic for the capacitated arc routing problem." European Journal of Operational Research 147, no. 3 (2003): 629-643.

[57] E. Benavent, V. Campos, A. Corberán, E. Mota, The capacitated arc routing problem: Lower bounds, Networks 22 (7) (1992) 669–690 (1992).

[58] S. Tilkov, S. Vinoski, Node.js: Using javascript to build high-performance network programs, IEEE Internet Computing 14 (6) (2010) 80–83 (2010).

[59] R. Dahl, et al., Node.js, in: URL: http://s3. amazonaws. com/four. livejournal/20091117/js conf. pdf, 2009, accessed: 2017-01-10.

[60] V. Agafonkin, Leaflet: An open-source javascript library for mobile-friendly interactive maps, http://leafletjs.com/ (2014).

[61] M. Bostock, V. Ogievetsky, J. Heer, D3: Data-driven documents, IEEE Transactions Visualization & Computer Graphics (Proceedings of the IEEE Information Visualization Conference) 17 (12) (2011) 2301–2309 (2011).

[62] P. Shannon, A. Markiel, O. Ozier, N. S. Baliga, J. T. Wang, D. Ramage, N. Amin, B. Schwikowski, T. Ideker, Cytoscape: A software environment for integrated models of biomolecular interaction networks, Genome Research 13 (11) (2003) 2498–2504 (2003).

[63] J. Resig, et al., Jquery, https://jquery.com/ (2006), accessed: 2017-01-10.

[64] M. Otto, J. Thornton, Bootstrap, http://getbootstrap.com/ (2013), accessed: 2017-02-22.

[65] qtip, https://github.com/cytoscape/cytoscape.js-qtip, accessed: 2017-02-15.

[66] Vex, https://github.com/HubSpot/vex, accessed: 2017-03-01.

[67] Alertify, https://github.com/MohammadYounes/AlertifyJS, accessed: 2017-02-17.

[68] V. Agafonkin, Interactive choropleth map, http://leafletjs.com/examples/choropleth/ (2015), accessed: 2017-02-15.

[69] J. Ashkenas, M. Bloch, S. Carter, A. Cox, The facebook offering: How it compares (2012). http://www.nytimes.com/interactive/2012/05/17/business/\dealbook/how-the-facebook-offering-compares.html accessed: 2017-02-15.

[70] M. E. Smoot, K. Ono, J. Ruscheinski, P.-L. Wang, T. Ideker, Cytoscape 2.8: New features for data integration and network visualization, Bioinformatics 27 (3) (2011) 431–432 (2011).

[71] S. Maere, K. Heymans, M. Kuiper, Bingo: A cytoscape plugin to assess overrepresentation of gene ontology categories in biological networks, Bioinformatics 21 (16) (2005) 3448–3449 (2005).

[72] M. S. Cline, M. Smoot, E. Cerami, A. Kuchinsky, N. Landys, C. Workman, R. Christmas, I. Avila-Campilo, M. Creech, B. Gross, et al., Integration of biological networks and gene expression data using cytoscape, Nature Protocols 2 (10) (2007) 2366–2382 (2007).

[73] J. Montojo, K. Zuberi, H. Rodriguez, F. Kazi, G. Wright, S. L. Donaldson, Q. Morris, G. D. Bader, Genemania cytoscape plugin: Fast gene function predictions on the desktop, Bioinformatics 26 (22) (2010) 2927–2928 (2010).

[74] M. Kucera, R. Isserlin, A. Arkhangorodsky, G. D. Bader, Autoannotate: A cytoscape app for summarizing networks with semantic annotations, F1000Research 5 (1717) (2016). doi:10.12688/f1000research.9090.1.

[75] V. Kofia, R. Isserlin, A. M. Buchan, G. D. Bader, Social network: a cytoscape app for visualizing co-authorship networks, F1000Research 4 (481) (2015). doi:10.12688/f1000research.6804.3.

[76] M. Franz, Tokyo railways, http://js.cytoscape.org/demos/f64e811fc3311414e083/ (2016), accessed 2017-04-03.

[77] E. A. Cabral, M. Gendreau, G. Ghiani, G. Laporte, Solving the hierarchical Chinese postman problem as a rural postman problem, European Journal of Operational Research 155 (1) (2004) 44–50 (2004).

[78] G. Ghiani, G. Improta, An algorithm for the hierarchical Chinese postman problem, Operations Research Letters 26 (1) (2000) 27–32 (2000).

[79] P. Korteweg, T. Volgenant, On the hierarchical Chinese postman problem with linear ordered classes, European Journal of Operational Research 169 (1) (2006) 41–52 (2006).

[80] P. W. Eklund, S. Kirkby, S. Pollitt, A dynamic multi-source dijkstra's algorithm for vehicle routing, in: L. C. Jain, V. L. Narasimhan (Eds.), Intelligent Information Systems, 1996., Australian and New Zealand Conference on Intelligent Information Systems, IEEE, Adelaide, Australia, 1996, pp. 329–333.

[81] A. Jotshi, Q. Gong, R. Batta, Dispatching and routing of emergency vehicles in disaster mitigation using data fusion, Socio-Economic Planning Sciences 43 (1) (2009) 1–24 (2009).

[82] P. Murray-Tuite, H. Mahmassani, Methodology for determining vulnerable links in a transportation network, Transportation Research Record: Journal of the Transportation Research Board (1882) (2004) 88–96 (2004).

[83] B. Balcik, B. M. Beamon, K. Smilowitz, Last mile distribution in humanitarian relief, Journal of Intelligent Transportation Systems 12 (2) (2008) 51–63 (2008).

[84] M. Jacomy, T. Venturini, S. Heymann, M. Bastian, Forceatlas2: A continuous graph layout algorithm for handy network visualization designed for the gephi software, PloS ONE 9 (6) (2014) e98679 (2014).

[85] C. Schulz, A. Nocaj, J. Goertler, O. Deussen, U. Brandes, D. Weiskopf, Probabilistic graph layout for uncertain network visualization, IEEE Transactions on Visualization and Computer Graphics 23 (1) (2017) 531–540 (2017).

[86] M. Xia, J. Wang, Y. He, Brainnet viewer: A network visualization tool for human brain connectomics, PloS ONE 8 (7) (2013) e68910 (2013).

[87] Afsar, H. Murat. "A branch-and-price algorithm for capacitated arc routing problem with flexible time windows." *Electronic Notes in Discrete Mathematics* 36 (2010): 319-326.

[88] Bodin, Lawrence D., and Samuel J. Kursh. "A computer-assisted system for the routing and scheduling of street sweepers." *Operations Research* 26:4 (1978): 525-537.

[89] Dror, Moshe, and Janny M.Y. Leung. "Combinatorial optimization in a cattle yard: Feed distribution, vehicle scheduling, lot sizing, and dynamic pen assignment." *Industrial Applications of Combinatorial Optimization*. Springer US, (1998): 142-171.

[90] Eglese, Richard W. "Routing winter gritting vehicles." *Discrete Applied Mathematics* 48:3 (1994): 231-244.

[91] Mullaseril, Paul Abraham. "Capacitated rural postman problem with time windows and split delivery." Ph.D. dissertation, University of Arizona, Tucson, AZ (1997).

[92] J. Nossack, Bruce L. Golden, Erwin Pesch, and Rui Zhang. "The windy rural postman problem with a time-dependent zigzag option. *European Journal of Operational Research* 258:3 (2017): 1131-1142.

[93] Edmonds, Jack, and Ellis L. Johnson. "Matching, Euler tours and the Chinese postman." *Mathematical Programming* 5:1 (1973): 88-124.

[94] Benavent, Enrique, Alessandro Carrotta, Ángel Corberán, José M. Sanchis, and Daniele Vigo. "Lower bounds and heuristics for the windy rural postman problem." *European Journal of Operational Research* 176:2 (2007): 855-869.

[95] Christofides, N., A. Mingozzi, and P. Toth. "An algorithm for time constrained travelling salesman problem. Technical Report, Imperial College, London (1981).

[96] Baker, Edward K. "Technical notean exact algorithm for the time-constrained traveling salesman problem." *Operations Research* 31:5 (1983): 938-945.

[97] Savelsbergh, Martin WP. "Local search in routing problems with time windows." *Annals of Operations Research* 4:1 (1985): 285-305.

[98] Golden, Bruce L., and Arjang A. Assad. "Vehicle routing with time-window constraints." *American Journal of Mathematical and Management Sciences* 6:3-4 (1986): 251-260.

[99] Solomon, Marius M., and Jacques Desrosiers. "Survey paper-time window constrained routing and scheduling problems." *Transportation Science* 22:1 (1988): 1-13.

[100] Bräysy, Olli, and Michel Gendreau. "Vehicle routing problem with time windows, Part I: Route construction and local search algorithms." *Transportation Science* 39:1 (2005): 104-118.

[101] Bräysy, Olli, and Michel Gendreau. "Vehicle routing problem with time windows, Part II: Metaheuristics." *Transportation Science* 39:1 (2005): 119-139.

[102] Aminu, U. F., and Richard W. Eglese. "A constraint programming approach to the Chinese postman problem with time windows." *Computers & Operations Research* 33:12 (2006): 3423-3431.

[103] Pearn, Wen-Lea, Arjang Assad, and Bruce L. Golden. "Transforming arc routing into node routing problems." *Computers & Operations Research* 14:4 (1987): 285-288.

[104] Reghioui, Mohamed, Christian Prins, and Nacima Labadi. "GRASP with path relinking for the capacitated arc routing problem with time windows." Edited by Mario Giacobini. *Workshops on Applications of Evolutionary Computation.* Springer Berlin Heidelberg (2007): 722-731.

[105] Irnich, Stefan. "A note on postman problems with zigzag service." *INFOR: Information Systems and Operational Research* 43:1 (2005): 33-39.

[106] Irnich, Stefan. "Solution of real-world postman problems." *European Journal of Operational Research* 190:1 (2008): 52-67.

[107] Solomon, Marius M. "Algorithms for the vehicle routing and scheduling problems with time window constraints." *Operations Research* 35:2 (1987): 254-265.

[108] Cardoso, Pedro JS, Gabriela Schütz, Andriy Mazayev, and Emanuel Ey. "Solutions in under 10 seconds for vehicle routing problems with time windows using commodity computers." Edited by Antnio Gaspar-Cunha, Carlos Henggeler Antunes, and Carlos Coello Coello. *Evolutionary Multi-Criterion Optimization.* Springer International Publishing (2015): 418-432.

[109] Thangiah, Sam R., Jean-Yves Potvin, and Tong Sun. "Heuristic approaches to vehicle routing with backhauls and time windows." *Computers & Operations Research* 23:11 (1996): 1043-1057.

[110] Lum, Oliver, Bruce Golden, and Edward Wasil, "OAR Lib: An open source arc routing library." Under review, 2017.

[111] Bräysy, Olli, and Geir Hasle, "Software tools and emerging technologies for vehicle routing and intermodal transportation." in *Vehicle Routing: Problems, Methods, and Applications* Ed. Paolo Toth and Daniele Vigo. SIAM - Society for Industrial and Applied Mathematics (2014): 351-380.

[112] M. Constantino, L. Gouveia, M.C. Mourão, and A.C. Nunes, The mixed capacitated arc routing problem with non-overlapping routes, European Journal of Operational Research, 244:2 (2015), 445-456.

[113] Lu, Quan, and Maged M. Dessouky, "A new insertion-based construction heuristic for solving the pickup and delivery problem with time windows." *European Journal of Operational Research* 175:2 (2006): 672-687.

[114] Matis, Peter, "Decision support system for solving the street routing problem." *Transport* 23:3 (2008): 230-235.

[115] Poot, Alexander, Goos Kant, and Albert Peter Marie Wagelmans, "A savings based method for real-life vehicle routing problems." *Journal of the Operational Research Society* 53:1 (2002): 57-68.

[116] Sahoo, Surya, Seongbae Kim, Byung-In Kim, Bob Kraas, and Alexander Popov Jr., "Routing optimization for waste management." *Interfaces* 35:1 (2005): 24-36.

[117] Tang, Hao, and Elise Miller-Hooks, "Interactive heuristic for practical vehicle routing problem with solution shape constraints." *Transportation Research Record: Journal of the Transportation Research Board* 1964:1 (2006): 9-18.

[118] B. Dussault, B. Golden, C. Groër, and E. Wasil, Plowing with precedence: A variant of the windy postman problem, Computers & Operations Research, 40:4 (2013), 1047-1059.

[119] E. Benavent, Á. Corberán, I. Plana, and J.M. Sanchis, Min-max k-vehicles windy rural postman problem, Networks 54:4 (2009), 216-226.

[120] E. Benavent, Á. Corberán, G. Desaulniers, F. Lessard, I. Plana, and J. M. Sanchis, A branch-price-and-cut method for the min-max k-vehicle windy rural postman problem, Networks 63:1 (2014), 34-45.

[121] E. Benavent, Á. Corberán, J.M. Sanchis, A metaheuristic for the min-max windy rural postman problem with k vehicles, Computational Management Science, 7:3 (2010), 269-287.

[122] E. Benavent, Á. Corberán, E. Piñana, I. Plana, and J.M. Sanchis, New heuristic algorithms for the windy rural postman problem, Computers & Operations Research 32:12 (2005), 3111-3128.

[123] Win, Zaw, "On the windy postman problem on Eulerian graphs." *Mathematical Programming* 44:1(1989): 97-112.

[124] G.N. Frederickson, M.S. Hecht, and C.E. Kim, Approximation algorithms for some routing problems, Foundations of Computer Science, 17th Annual Symposium *IEEE* (1976), 216-227.

[125] W.L. Pearn, Solvable cases of the k-person Chinese postman problem, Operations Research Letters 16:4 (1994), 241-244.

[126] D. Ahr, and G. Reinelt, "New heuristics and lower bounds for the min-max k-Chinese postman problem," Algorithms—ESA 2002, R. Mohring, R. Raman (Editors), Springer Berlin Heidelberg, 2002, Vol. 2461, pp. 64-74.

[127] C. Prins, A simple and effective evolutionary algorithm for the vehicle routing problem, Computers & Operations Research 31:12 (2004), 1985-2002.

[128] G. Karypis, and V. Kumar, METIS-unstructured graph partitioning and sparse matrix ordering system, version 2.0, webpage. http://glaros.dtc.umn.edu/gkhome/metis/metis/overview, 1995.

[129] G. Karypis, and V. Kumar, A fast and high quality multilevel scheme for partitioning irregular graphs, SIAM Journal on Scientific Computing 20:1 (1998), 359-392.

[130] B. Kim, S. Kim, and S. Sahoo, Waste collection vehicle routing problem with time windows, Computers & Operations Research 33:12 (2006), 3624-3642.

[131] He, Ruhan, Weibin Xu, Jiaxia Sun, and Bingqiao Zu, "Balanced k-means algorithm for partitioning areas in large-scale vehicle routing problem." *Intelligent Information Technology Application, 2009. IITA 2009. Third International Symposium* Vol. 3. IEEE (2009): 87-90.

[132] J. Kalcsics, "Districting problems," Location Science, G. Laporte, S. Nickel, and F. Saldanha da Gama (Editors), Springer International Publishing, 2015, pp. 595-622.

[133] M. López-Ibánez, J. Dubois-Lacoste, T. Stützle, and M. Birattari, The irace package: Iterated racing for automatic algorithm configuration, Technical Report TR/IRIDIA/2011-004, IRIDIA, Université Libre de Bruxelles, Belgium, 2011.

[134] Lum, O., Cerrone, C., Golden, B., and Wasil, E. (2017). Partitioning a street network into compact, balanced, and visually appealing routes. *Networks*, 69(3), 290-303.

[135] Bara'a, A. Attea, Enan A. Khalil, and Ahmet Cosar, "Multi-objective evolutionary routing protocol for efficient coverage in mobile sensor networks." *Soft Computing* 19:10 (2015): 2983-2995.

[136] Magaia, Naércio, Nuno Horta, Rui Neves, Paulo Rogério Pereira, and Miguel Correia, "A multi-objective routing algorithm for wireless multimedia sensor networks." *Applied Soft Computing* 30 (2015): 104-112.

[137] Rahat, Alma As-Aad Mohammad, Richard M. Everson, and Jonathan E. Fieldsend, "Multi-objective routing optimisation for battery-powered wireless sensor mesh networks." *Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation,* ACM, 2014.

[138] Gulczynski, Damon, Bruce Golden, and Edward Wasil, "The period vehicle routing problem: New heuristics and real-world variants." *Transportation Research Part E: Logistics and Transportation Review* 47:5 (2011): 648-668.

[139] Mandal, Santosh Kumar, Dario Pacciarelli, Arne Lkketangen, and Geir Hasle, "A memetic NSGA-II for the bi-objective mixed capacitated general routing problem." *Journal of Heuristics* 21:3 (2015): 359-390.

[140] Janssens, Jochen, Joos Van den Bergh, Kenneth Sörensen, and Dirk Cattrysse, "Multi-objective microzone-based vehicle routing for courier companies: From

tactical to operational planning." *European Journal of Operational Research* 242:1 (2015): 222-231.

[141] Ombuki, Beatrice, Brian J. Ross, and Franklin Hanshar, "Multi-objective genetic algorithms for vehicle routing problem with time windows." *Applied Intelligence* 24:1 (2006): 17-30.

[142] Baños, Raúl, Julio Ortega, Consolación Gil, Antonio L. Márquez, and Francisco De Toro, "A hybrid meta-heuristic for multi-objective vehicle routing problems with time windows." *Computers & Industrial Engineering* 65:2 (2013): 286-296.

[143] Melián-Batista, B., De Santiago, A., AngelBello, F. and Alvarez, A., "A bi-objective vehicle routing problem with time windows: a real case in Tenerife." *Applied Soft Computing* 17 (2014): 140-152.

[144] Benavent, Enrique, Ángel Corberán, Isaac Plana, and José M. Sanchis, "New Facets and an enhanced branch-and-cut for the min-max k-vehicles windy rural postman Problem." *Networks* 58 (2011), 255-272.

[145] Cordeau, Jean-François, "A branch-and-cut algorithm for the dial-a-ride problem." *Operations Research* 54:3 (2006): 573-586.

[146] Bode, Claudia, and Stefan Irnich. "Cut-first branch-and-price-second for the capacitated arc-routing problem." *Operations Research* 60:5 (2012): 1167-1182.

[147] Ahr, D., and Reinelt, G. (2002). New heuristics and lower bounds for the min-max k-chinese postman problem. In *AlgorithmsESA 2002* (pp. 64-74). Springer Berlin Heidelberg.

[148] Derigs, U. *Optimization and Operations Research*. Eolss Publishers Company Limited, 2009.

[149] Dussault, B., Golden, B., Groër, C., and Wasil, E. (2013). Plowing with precedence: A variant of the windy postman problem. *Computers & Operations Research*, 40(4), 1047-1059.

[150] Karypis, G., and Kumar, V. (1998). A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on Scientific Computing*, 20(1), 359-392.

[151] Letchford, A. N., Reinelt, G., and Theis, D. O. (2004). A faster exact separation algorithm for blossom inequalities. In *Integer Programming and Combinatorial Optimization* (pp. 196-205). Springer Berlin Heidelberg.

[152] Lum, O., Cerrone, C., Golden, B., and Wasil, E. (2017). Partitioning a street network into compact, balanced, and visually appealing routes. *Networks*, 69(3), 290-303.

[153] Padberg, M. W., and Rao, M. R. (1982). Odd minimum cut-sets and b-matchings. *Mathematics of Operations Research*, 7(1), 67-80.

[154] G.N. Frederickson, Approximation algorithms for some postman problems, Journal of the ACM, 26:3 (1979), 538-554.

[155] M.W. Padberg, and M.R. Rao, Odd minimum cut-sets and b-matchings, Mathematics of Operations Research 7:1 (1982), 67-80.

[156] Cohon, Jared L., *Multiobjective programming and planning.* Courier Corporation, 2013.